

NAVAL POSTGRADUATE SCHOOL

Monterey, California



19971124 006

THESIS

DTIC QUALITY INSPECTED 2

**USING ARTIFICIAL NEURAL NETWORKS TO IDENTIFY
UNEXPLODED ORDNANCE**

by

Jeffrey A. May

June 1997

Thesis Advisor:

Nelson D. Ludlow

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
June 1997

3. REPORT TYPE AND DATES COVERED
Master's Thesis

4. TITLE AND SUBTITLE
USING ARTIFICIAL NEURAL NETWORKS TO IDENTIFY UNEXPLODED ORDNANCE

5. FUNDING NUMBERS

6. AUTHOR(S)
May, Jeffrey A.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION
REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING
AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

The clearing of unexploded ordnance (UXO) is a deadly and time consuming process. The U.S. Government is currently spending millions of dollars to remove UXO's from bases that are closing around the world. Existing methods for detecting UXO's only inform the clearing team that a piece of metal is present, rather than the type of metal, either UXO, shrapnel, or garbage. A lot of time and money is spent digging up every piece of metal detected. This thesis presents the use of artificial neural networks to determine the type of UXO that is detected. A multi-layered feed-forward neural network using the back propagation training algorithm was developed using the language Lisp. The network was trained to recognize five pieces of ammunition. Results from the research show that four out of five pieces of ammunition from the test set were identified with an accuracy of .99 out of 1.0. The network also correctly identified that a tin can was not one of the five pieces of ammunition.

14. SUBJECT TERMS
Unexploded Ordnance, Artificial Neural Networks

15. NUMBER OF PAGES
137

16. PRICE CODE

17. SECURITY
CLASSIFICATION OF REPORT
Unclassified

18. SECURITY
CLASSIFICATION OF THIS
PAGE
Unclassified

19. SECURITY
CLASSIFICATION OF
ABSTRACT
Unclassified

20. LIMITATION OF
ABSTRACT

UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18

Approved for public release; distribution is unlimited

**USING ARTIFICIAL NEURAL NETWORKS TO
IDENTIFY UNEXPLODED ORDNANCE**

Jeffrey A. May
Captain, United States Army
B.S., Creighton University, 1988

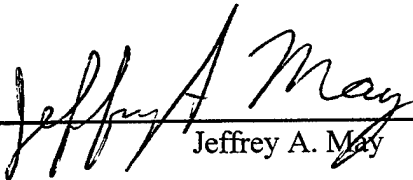
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

**NAVAL POSTGRADUATE SCHOOL
June 1997**

Author


Jeffrey A. May


Approved by:



Nelson D. Ludlow, Thesis Advisor



Robert B. McGhee, Second Reader



Ted Lewis, Chairman
Department of Computer Science

ABSTRACT

The clearing of unexploded ordnance (UXO) is a deadly and time consuming process. The U.S. Government is currently spending millions of dollars to remove UXO's from bases that are closing around the world. Existing methods for detecting UXO's only inform the clearing team that a piece of metal is present, rather than the type of metal, either UXO, shrapnel, or garbage. A lot of time and money is spent digging up every piece of metal detected. This thesis presents the use of artificial neural networks to determine the type of UXO that is detected. A multi-layered feed-forward neural network using the back propagation training algorithm was developed using the language Lisp. The network was trained to recognize five pieces of ammunition. Results from the research show that four out of five pieces of ammunition from the test set were identified with an accuracy of .99 out of 1.0. The network also correctly identified that a tin can was not one of the five pieces of ammunition.

TABLE OF CONTENTS

I. INTRODUCTION.....	1
A. GOALS	1
B. BACKGROUND AND MOTIVATION.....	1
C. RESEARCH QUESTIONS.....	2
D. ORGANIZATION	2
II. BACKGROUND.....	3
A. INTRODUCTION.....	3
B. UNEXPLODED ORDNANCE IDENTIFICATION	3
1. Traditional Methods.....	3
2. Existing Methods	4
3. Artificial Neural Network Approach	6
C. ARTIFICIAL NEURAL NETWORKS	6
1. The Biological Neuron.....	7
2. The Artificial Neuron.....	9
3. Artificial Neural Networks	10
4. Learning.....	12
5. Network Architecture	15
D. SUMMARY	18
III. ARTIFICIAL NEURAL NETWORK DESIGN	19
A. CHOOSING A NETWORK DESIGN.....	19
1. Design Factors	19
a. Speed of Execution.....	19
b. Generalization.....	20
c. Scalability	20
d. Learning Speed.....	20
e. Number of Layers.....	21
f. Connectivity	22
2. Choosing an Architecture	23
B. MULTI-LAYERED FEED-FORWARD NETWORK	23
C. BACK PROPAGATION ALGORITHM	27
IV. AMMUNITION NETWORK.....	35
A. INTRODUCTION.....	35
B. AMMUNITION TYPES	35
C. DATA PREPARATION	38
1. Magnetometer	38
2. Data Collection	39
D. LISP IMPLEMENTATION OF NETWORK	40
1. Neuron Class.....	41
2. Layer Class	42
3. Network Class.....	44
4. Back Propagation Algorithm	46
5. User Interface.....	50
V. RESULTS.....	53
A. AMMUNITION GRAPHS.....	53
B. TRAINING	56
C. TESTING.....	57
VI. CONCLUSIONS	61

A. THESIS QUESTIONS	61
B. LESSONS LEARNED	63
C. RECOMMENDATIONS FOR FUTURE RESEARCH.....	64
APPENDIX A: SOURCE CODE (NEURON CLASS).....	65
APPENDIX B: SOURCE CODE (LAYER CLASS)	69
APPENDIX C: SOURCE CODE (NETWORK CLASS).....	73
APPENDIX D: SOURCE CODE (BACK PROPAGATION).....	77
APPENDIX E: SOURCE CODE (AMMO RECOGNITION)	85
APPENDIX F: INPUT DATA	95
APPENDIX G: SOURCE CODE (C++ NEURAL NETWORK).....	111
LIST OF REFERENCES.....	123
INITIAL DISTRIBUTION LIST	125

ACKNOWLEDGEMENTS

This research was possible due to the efforts of many people. Most notably is that of my thesis advisor and second reader. Major Nelson Ludlow's mentorship was truly instrumental in my understanding of artificial neural networks. My ability to develop the neural network using object oriented programming in Lisp is a credit to Professor Robert McGhee's teaching.

Sincere thanks go out to three fellow students. Steve Weldon and Ken Fritzsche sacrificed many hours of their valuable time to help me gather data on the unexploded ordnance training sets and test sets. Paul Arcangeli was instrumental in providing information on current techniques of UXO removal and in choosing the training set. Their selfless efforts to help me achieve many of my research goals are a tribute to the spirit of academics. I would also like to thank the 87th EOD for loaning me the ammunition which made up the training sets.

I would also like to thank the students and faculty members at the Naval Postgraduate School. I have found the educational experience to be a positive one and this is only possible with the outstanding faculty and peers with which this institution is blessed.

Finally, I am grateful to my wife Angela and sons Matthew and Garrett for their patience, understanding, and support throughout this research and my studies.

I. INTRODUCTION

A. GOALS

The goal of this thesis is to determine if an artificial neural network is capable of identifying unexploded ordnance. The intent is to develop an artificial neural network to correctly identify a limited set of unexploded ordnance. A successful neural network will aid in the clearing of United States military bases by identifying those detection's that are unexploded ordnance and should be excavated by expert EOD personnel, from those that are not unexploded ordnance and can be removed by less trained personnel.

B. BACKGROUND AND MOTIVATION

As the military continues to scale down, the job of turning the land over to the civilian sector is a labor intensive process. One of the biggest problems is the clearing of unexploded ordnance (UXO) from the bases. It is a time consuming and expensive job to dig up every piece of metal that returns a signal on the detection device. The metal detected could be an actual round or fragments from exploded rounds as well as junk that may be in the area.

The Department of Defense has recently approved two organizational structures to confront the challenge of UXO remediation and wide-area de-mining. The objective of the first committee is to develop a fully coordinated requirements driven research and development program for countermining, de-mining, site remediation, range clearance, and explosive ordnance disposal. Within the first committee there is a specific group focused on detection technology. The second committee will focus on current technologies and ways to improve in the future. One of the phases will examine current UXO remediation, active range UXO clearance and explosive ordnance disposal efforts. So as you can see the UXO problem is real and getting a lot of attention in today's military.

At the Naval Postgraduate School a team has been put together to develop an autonomous vehicle or robot, which will survey the area for UXO's. The autonomous

vehicle, called Shepherd, is well under way. Shepherd is a four wheel independent steering, autonomous vehicle. The four wheel independent steering allows Shepherd a high level of mobility. The vehicle needs the means to locate and classify various unexploded ordnance via standard sensors, such as a magnetometer. This is the basis of my thesis.

C. RESEARCH QUESTIONS

This thesis will examine the following research areas:

- Are artificial neural networks able to correctly identify, within a certain degree of precision, various types of unexploded ordnance both surface laid and buried?
- What type of neural network architecture is best for the job?
- What is the training set to be used in the training of the neural network?
- With what precision are the objects correctly identified?

D. ORGANIZATION

Chapter II provides a general overview of traditional and current techniques for identifying unexploded ordnance and gives an introduction to the artificial neuron, types of neural networks and training methods. Chapter III covers the process of choosing a network architecture and an in depth discussion of the feed-forward and back-propagation algorithm used for the neural network. Chapter IV presents my artificial neural network design for the UXO project as well as what ammunition was used and how the data was gathered. In Chapter V the results of testing the neural network are presented, and Chapter VI summarizes the thesis.

II. BACKGROUND

A. INTRODUCTION

The old saying that "time is money" holds true for the clearing of unexploded ordnance. The methods used to accomplish range remediation, both in the past and recent, are time consuming. It takes time to dig up every piece of metal that returns a signal. This time costs the government a lot of money. If we can identify objects that are ordnance from objects that are not, such as tin can or shrapnel, we can remove objects safely. To EOD personnel, every piece of unexploded ordnance on the ground is potentially deadly and great caution is taken to avoid injury. Being able to determine unexploded ordnance from junk can save lives as well as time and money.

In this thesis the term unexploded ordnance refers to projectiles, either tube launched or rocket assisted, bombs dropped from aircraft, and thrown ammunition, such as hand grenades. Excluded from this list are land mines. Land mines present a whole different challenge to range clearing efforts. In this chapter the techniques used to clear unexploded ordnance are addressed. Then, the fundamental of artificial neural networks will be introduced in order to provide the reader with the background necessary to make the research more understandable.

B. UNEXPLODED ORDNANCE IDENTIFICATION

1. Traditional Methods

The traditional method of clearing unexploded ordnance consists of personnel using some type of metal detector to detect the general location of ordnance and then marking the spot with a flag. The next step is digging up the ordnance and removing it from the site. As you can tell, a lot of unnecessary metal is dug up and a lot of time is wasted. This method is both dangerous and expensive. However, for many years it was the only method available. With the massive number of acres of land from closed bases

needing cleared and an ever decreasing defense budget, new methods of UXO removal must be developed.

2. Existing Methods

The devices for detecting unexploded ordnance have improved over the years. For the most part however, the methods for clearing UXO's have not. Many government contractors are still using the traditional method explained above. The most common technique used to deter the cost of sweeping an entire base is some form of sampling. Areas to be cleared are assigned grid zone designators. The size of a grid may vary depending on the terrain, but a 100 x 100 foot grid is a good starting place. This grid is then broken down into sub-grids or lanes that are randomly chosen for sweeping with a magnetometer. Any detection's are flagged for removal. The metal removed is classified as either a UXO, shrapnel, or trash. A determination on whether to clear an area is made based upon, among other things, the concentration of UXO classified objects in the sub-grid or lane.

Some of the other factors taken into account are the history of the area, what will the land be used for, and location of the land. The history of a grid must be investigated before one can be chosen. The history looks at the type of rounds fired into the area, where the rounds were fired from, and where they were to impact, as well as a margin of error based on the capabilities of the round. The capabilities of the rounds also include the penetrating depth of the round. This way the clearing team knows up to what depth to clear. What the land will be used for also determines the clearing depth. Construction in the area may only penetrate to a certain depth, therefore clearance beyond that depth is wasting time and money. The location of the land also plays an important role. If the land is in a residential area, the number of sub-grids or lanes sampled may be greater than that of a remote area with thick brush that will not be used for anything in the near future.

Recent statistical computer models have aided in the grid zone technique by allowing the clearing team to randomly sample the detection's within a sub-grid or lane.

The models compute concentration of UXO's based on a formula and the data from the random samples. A computer aided tool known as the Ordnance and Explosives Knowledge Base (OE-KB) is being developed by the U.S. Army Engineer Center in order to build a knowledge base on the characteristics of detection's [Ref. 1]. Some of these characteristics include the sensor used, the type of round, the depth, the angle, and the type of soil. Detection readings are fed into the database in order to attempt to determine the type and depth of the ammunition. OE-KB uses sophisticated mathematical algorithms, computerized pattern-recognition, and data fusion (the combination and comparison of data-sets from two or more different types of geophysical instruments) to help differentiate between munitions and non-munitions and estimate depth of burial [Ref. 1].

The most common type of metal detector used is the magnetometer. A magnetometer was used to gather the input data for the neural network in this thesis. The GA-72Cd Magnetic Locator, made by Schonstedt Instruments Company, detects the magnetic field of iron and steel objects [Ref. 2]. The magnetometer has an audio and digital output, and a polarity indicator to help pinpoint the target and even determine its orientation. Tests have shown that magnetometers detect ferrous munitions and are effective to depths of 2 or 3 meters [Ref. 1].

In a paper published by Matthew Gifford and John E. Foley for Sanford Cohen and Associates Inc., a neural network is used to identify the weight and depth of the UXO [Ref. 3]. A dig team recorded data obtained from a Geonics EM-61 pulsed induction sensor. This information was then fed into a multi-layered feed-forward neural network to determine the weight and depth of the object. The network used back-propagation to train on 107 items and then was tested on 40. The outcome was a 77% detection rate with a cost savings of 74% over the Amag and flag technique. This study seems to prove that the use of a neural network to classify and reduce UXO remediation is a valid option. The problem with this research is that any piece of metal that has a similar mass as an UXO and is at the appropriate depth will be dug up.

3. Artificial Neural Network Approach

An artificial neural network is good for pattern recognition and classification problems. With the use of a magnetometer, UXO's produce a signal that varies across a grid. The grid can be fed into a neural network. This pattern of signals can be classified by the neural network as a type of UXO that the network has been trained to recognize. That is the goal of this thesis. Given a grid of inputs, say 60 X 60 cm, take these values returned by the magnetometer and train a network to recognize certain patterns that relate to several types of UXO's. If successful, the artificial neural network will be able to identify UXO's that need to be removed versus returns from a magnetometer that could just be scrap metal. This would save a lot of time and money. To understand how an artificial neural network can accomplish this mission, the following section will give an overview of how an artificial neural network works.

C. ARTIFICIAL NEURAL NETWORKS

What are artificial neural networks and why are they used? Artificial neural networks are an attempt to make computers use the same reasoning as humans. Computers use the Von Neumann architecture that is very efficient for number crunching. Computers greatly out perform humans in this area and hence, their popularity. When it comes to perceptual problems, humans are way out in front of computers. Table 2.1 points out some of the key differences between computers and the human brain. Why use an artificial neural network? Neural networks are a good solution to a problem that is not well defined. If the input data may vary for the same problem, then a neural network is much more forgiving than a traditional algorithm. A neural network also promotes the ability to use parallel processing. From a purely scientific stand point, one reason to use a neural network is an attempt to model the human decision making process. The key to human success lies in the biological neuron.

	<i>Von Neumann Computer</i>	<i>Biological Neural System</i>
Processor	Complex High speed One or a few	Simple Low speed A large number
Memory	Separate from a processor Localized Non-content addressable	Integrated into processor Distributed Content addressable
Computing	Centralized Sequential Stored programs	Distributed Parallel Self-learning
Reliability	Very vulnerable	Robust
Expertise	Numerical & symbolic manipulation	Perceptual problems
Operating Environment	Well defined Well constrained	Poorly defined Unconstrained

Table 2.1: Von Neumann computer versus biological neural system [Ref. 4].

1. The Biological Neuron

A neuron is a cell that processes information in humans. Figure 2.1 shows a picture of a neuron and all of its components. The neuron is made up of a cell body (soma), branch like Figures called dendrites and an axon which also protrudes from the soma. The dendrites are the receivers of signals from other neurons while the axon is the transmitter of signals to other neurons. The synapses are between the dendrites of one neuron and the axon of another neuron. The synapses release neurotransmitters that,

depending on their type, can excite or inhibit the signal. The synapses are the key to learning in the neuron. They can be adjusted based on their experience.

In order to understand the magnitude and complexity of the neurons in a human, the cerebral cortex is examined. The cerebral cortex is about 2 to 3 millimeters thick with a surface area of about 2,200 cm², about twice the size of a standard computer keyboard [Ref. 4]. The cerebral cortex contains about 10¹¹ neurons, which is approximately the number of stars in the Milky Way [Ref. 4]. Each neuron is connected to approximately 10³ other neurons and the human brain contains roughly 10¹⁴ to 10¹⁵ interconnections [Ref. 4].

The neuron operates at a speed of a few milliseconds. From the time it takes a human to recognize an object, it has been determined that the perceptual decisions cannot take more than 100 or so serial steps [Ref. 4]. Therefore, the brain must run parallel processes that are about 100 steps long for such a task. In the same research it was shown that only a very small amount of information was transferred in this time. Therefore it is believed that critical information is not transmitted directly, but captured and distributed in the interconnections [Ref. 4]. As you will see, this is the basic idea behind the artificial neural network.

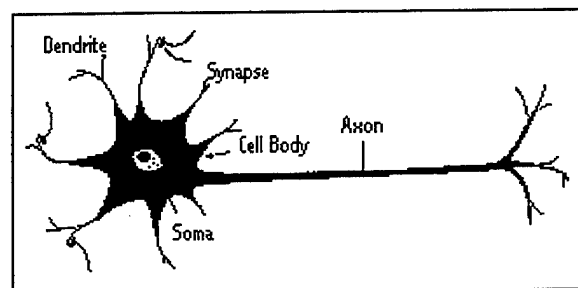


Figure 2.1: A biological neuron.

2. The Artificial Neuron

The artificial neuron attempts to model the biological neuron using a computer.

Figure 2.2 shows the layout of an artificial neuron. There are three basic components:

- The *synapse* is modeled by a *weight* associated with that connection to the neuron and an *input signal* from the previous neuron or source. Here, the weight and the input signal are multiplied to provide an input value for that connection. The key to a successful neural network lies in the value of the weights associated with each neuron. Chapter IV will discuss how the back-propagation algorithm assigns these weights.
- The *dendrite* provides the input from the synapse to the soma. This is the connection leading into the neuron. Here the artificial neuron has an *adder* which computes the summation of the weighted inputs from all the synapses.
- The *soma* is modeled by an *activation function* for limiting the amplitude of the output signal from the neuron. The activation function is a nonlinear function. Four typical types of activation functions are the *threshold*, *piecewise linear*, *sigmoid* and *Gaussian* functions shown in Figure 2.3. The *threshold* function is an on-off type of function. This means the neuron will only fire at the vertical on the graph. The *piecewise linear* function displays a little better firing distribution. *Sigmoid* functions, such as the *logistics* function, are the most widely used activation functions. They exhibit a strictly increasing function that provides the desired asymptotic properties [Ref. 4]. By inspection, it is easy to see that the *Gaussian* function is increasing the intensity with the higher negative weights and decreasing intensity with the higher positive weights. The output range of the neuron is a closed interval $[-1, 1]$.

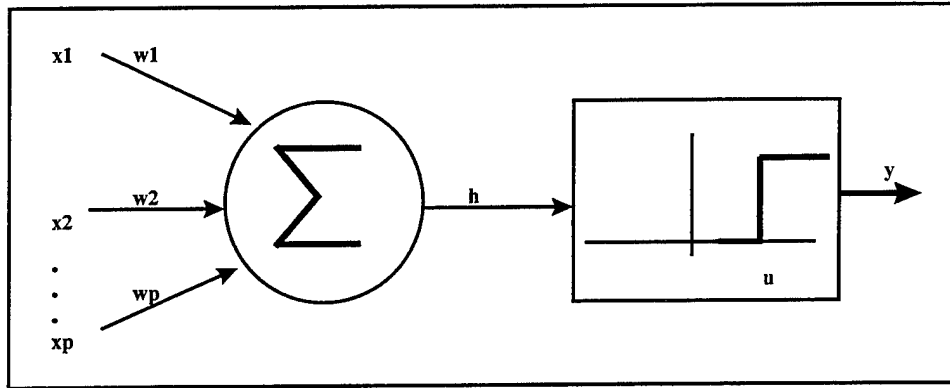


Figure 2.2: McCulloch-Pitts model of a neuron [Ref. 4].

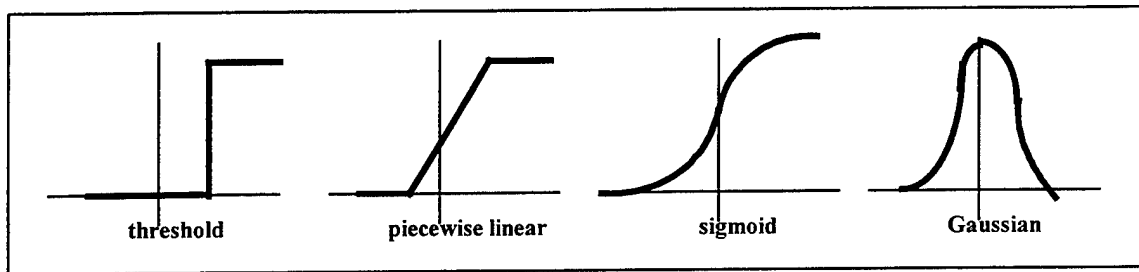


Figure 2.3: Activation Functions.

A neuron can be described in mathematical terms by the following equation:

$$y_k = \phi \left(\sum_{j=1}^p w_{jk} x_j \right)$$

Where ϕ is the activation function, w_{jk} is the specific weights associated with that neuron and x_j is the signal input for that neuron. The value y_k is the value that is produced by that particular neuron which should be between 0.0 and 1.0. This value is then passed on to other neurons as the new input and the process starts all over again. This is the basic idea of how a neural network is formed.

3. Artificial Neural Networks

The way in which one combines neurons to produce an answer to a problem is called an artificial neural network. The number of input values determines the number of input neurons. There are several ways you can connect the next layer, or *hidden layer*, neurons to the input layer neurons. Several layers of neurons can be used. The number

of output neurons depends on the number of possible outcomes for a particular problem. The trick becomes knowing how many neurons to have in the hidden layer and the number of hidden layers to have in a network.

The way a neural network solves a problem can be traced to the idea of linear separability. Figure 2.4 shows the graph of an AND function and the line that is drawn by the neural network [Ref. 5]. The correct answer has been separated out from the wrong answers by the network. Notice that everything above the line will exhibit characteristics of the correct answer. The equation of a line is $Ax + By + C = 0$. This equation can be translated into the following equation from the AND neural network:

$$x_1w_1 + x_2w_2 + B = 0$$

where x_1 and x_2 are the input signals 1 and 0, and w_1 and w_2 are the weights associated with these neurons. B is a bias that is thrown in to help separate the problem.

When a problem is linearly inseparable, then multiple layers are needed to separate out the answer with multiple lines. Figure 2.6 shows the Exclusive-OR problem. Notice that two lines are needed to separate out the answer in this problem. The first hidden layer separates out the two correct answers. In this case, the output layer combines the correct outputs and separates true from false with a single line. By adding more layers the correct responses become more defined. However, it is argued that any problem can be solved with a three layer network [Ref. 6].

A three layer network is defined by the input layer, consisting of the input values, the hidden layer, consisting of neurons that separate the problem, and an output layer, consisting of neurons that produce the desired answers. The desired answer is defined by the training set. Let's say your neural network is designed for character recognition. The number of possible outputs is 26, one for each letter in the alphabet. Therefore, your output layer will have 26 neurons, each providing a value for the character that it represents. This value should be high for the neuron associated with the correct input and low for the rest of the neurons. For example, if the character 'C' is given, it should

produce a high output (close to 1.0) for output C, and a very low output (close to 0.0) for the other 25 outputs.

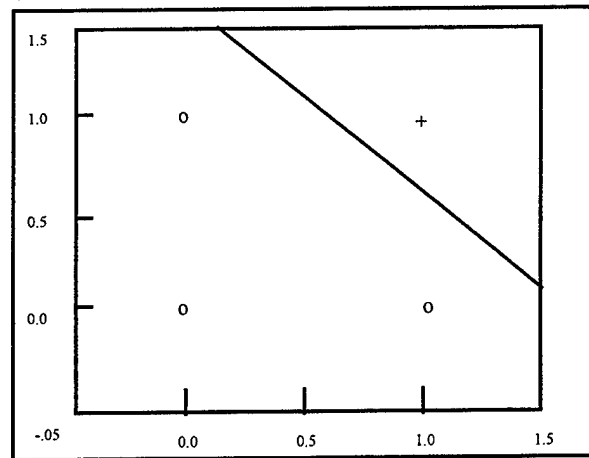


Figure 2.4: Classification line for logical AND[Ref. 5].

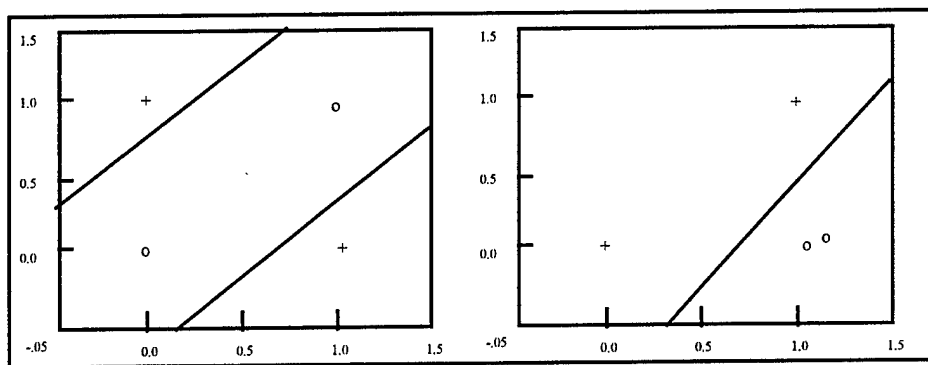


Figure 2.5: The figure on the left shows how the hidden layer separates the inputs. The figure on the right shows how the output layer makes the problem linear separable [Ref. 5].

4. Learning

There are three types of learning paradigms when it comes to neural networks:

- Supervised learning separates the training from the environment. The network must be taught to recognize the inputs for which it has been designed to produce an output. This is accomplished by developing a training set of samples of the environment for which the network is to perform. The network is then trained in an iterative process, via a training algorithm, until the

appropriate response is produced to the entire training set. The appropriate response is defined by the designer as an acceptable level of error. The means in which to reach the acceptable error is known as the steepest descent problem. Chapter III will go into great detail on the steepest descent problem. Once the designer is satisfied, the training algorithm is removed from the network. Now the network is tested on the environment. The network is now unsupervised and working on its own with an embedded knowledge from the training algorithm.

- Unsupervised learning is the exact opposite of supervised learning. There is no teacher. The network finds correlations between patterns in the input data and groups those correlations into categories. By comparing new input data to previous ones, the network can categorize the data and produce an answer.
- Reinforcement learning has no teacher to give direction to the answer. It must probe the environment to gain knowledge of the direction to travel in order to obtain the correct answer.

There are four learning rules that span the above learning paradigms:

- Error-correction rules basically perform as stated above in the supervised learning paradigm example. The weights in the network are adjusted in accordance with an output error. The desired output is subtracted from the actual output and this value is called the error for that run through the neural network. All of the weights are adjusted to decrease this error by a predetermined value each time through the network. Mathematically speaking

$$e_k = d_k - a_k, \quad \Delta w_{jk} = \beta * e * i, \quad w_{jk} = w_{jk} + \Delta w_{jk}$$

given k = neuron, w = weight, i = input value, β = predetermined step value, e = error, d = the desired value, and a = the actual value.

- Boltzmann learning applies to Boltzmann machines in which the neurons operate in a binary manner, +1 for the on state and -1 for the off state [Ref. 4]. The neurons are further divided into visible neurons, which interact with the

environment, and hidden neurons which do not. The machine itself operates in two modes: clamped, in which the visible neurons are set in their current states determined by the environment, and free, in which both the visible and hidden neurons are allowed to operate freely. The objective of Boltzmann learning is to adjust the connection weights so that the states of the visible units satisfy a particular probability. Weight changes are denoted by

$$\Delta w = \beta(\bar{p}_{jk} - p_{jk})$$

where β is the learning rate, and \bar{p}_{jk}, p_{jk} are the correlation's between the states of units j and k when the network operates in the clamped and free mode, respectively [Ref. 4].

- Hebbian rule is based on the fact that if two neurons are activated synchronously then the weight is strengthened. If the two neurons are activated asynchronously then the weight is weakened.

$$\Delta w_{jk} = \beta * o_j * o_k, \quad w_{jk} = w_{jk} + \Delta w_{jk}$$

where w_{jk} is the weight from neuron j to neuron k , o is the output of the neuron, and β is the learning rate. This learning rule is highly dependent on a recurring structure which will be discussed later.

- Competitive learning systems group the input data into categories. Based on the input, the correct group is stimulated and only those units fire. This is known as the winner-takes-all method. The output of the winner is set to 1 and all other nodes are set to 0. Only the weights of those that fire are updated. The good weight values from the input vectors of the winner are distributed equally over the entire set of input weights associated with the winning neuron.

$$\sum_k w_{jk} = 1, \quad \Delta w_{jk} = \begin{cases} \beta(x_k - w_{jk}) \\ 0 \end{cases}$$

Where w_{jk} is the weight, j is the neuron the weight is coming from, k is the neuron, x_k is the input value, and β is the learning rate. This has the overall effect of moving the weight vector of the winning neuron toward x [Ref. 7].

5. Network Architecture

There are many types of neural network architectures. Figure 2.6 provides a layout of how they are broken down. The recurrent networks are ones in which learning is based on associative memory. The loops that occur in the network act as a state that the network enters based on the input. Recurrent networks are dynamics systems. Each neuron is dependent on it's previous output due to the feedback nature of the network. In this manner they are also able to store information. The feed-forward networks contain no loops. They have no memory. This means that the value produced does not rely on previous inputs. The learning is supervised because it is based on the training set. Feed-forward networks are static in the sense that once they have been trained, they produce one set of output values for one set of input values. I will briefly discuss the types of networks presented in Figure 2.6 that have not already been discussed in the learning section.

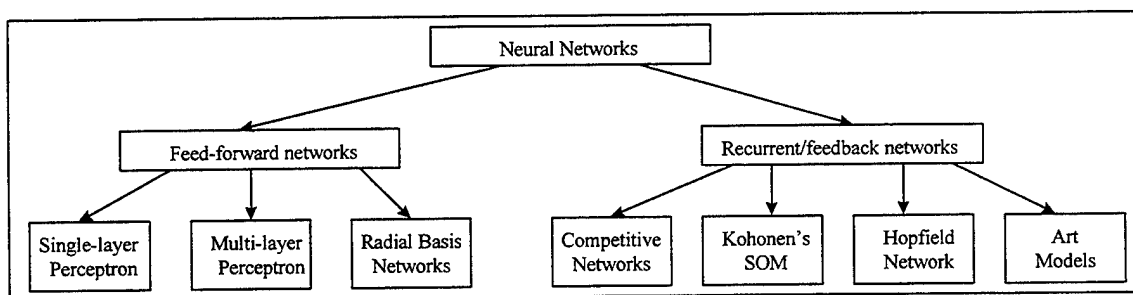


Figure: 2.6: A taxonomy of feed-forward and recurrent/feedback networks [Ref. 4].

This thesis concentrates on the feed-forward type of neural networks. The most widely used type of feed-forward network is the multi-layered feed-forward neural network. The performance of this network is exactly as the name implies. The input

signals are fed into the network and multiplied by the weights assigned to the signal. The weighted sum of each neuron is then fed to an activation function which produces the output value of the neuron. These values are then fed to the next layer and the process is repeated until there are no more layers. The neuron in the output layer with the greatest value represents the pattern that was fed into the network. The whole key to this network are the values of the weights. The algorithm used to adjust the weights is the back-propagation algorithm.

The radial basis functions (RBF) networks are a special form of the multi-layered feed-forward network with two layers, a hidden layer and an output layer. The hidden layer is a nonlinear mapping from the input values to the hidden layer. A nonlinear radial basis function is used as an activation function in the hidden layer. A linear combination of RBFs is used to convert the hidden layer to the linear output layer. The RBF has the following property:

$$F(x) = w_i \varphi(\|x - x_i\|)$$

where $\{\varphi(\|x - x_i\|) \mid i = 1, 2, \dots, N\}$ is a set of N arbitrary (generally nonlinear) RBF functions and $\| \cdot \|$ denotes a norm that is usually taken to be Euclidean [Ref. 7].

RBF networks are used for function approximation, pattern classification, prediction and control problems.

The Kohonen's Self Organizing Map (SOM) artificial neural network models the fact that in the biological neural network environment tends to strengthen connections that are close in physical proximity. The SOM model uses a two dimensional output grid of neurons that are connected to their neighbors. All of the inputs are connected to each neuron in the 2-dimensional grid. Figure 2.7 shows the set-up of a Kohonen SOM. The sum of each set of input weights from any input node total one. As the inputs are fed to the network, the neuron with the maximum sum of the weight values wins. This neuron's weight values are updated as are the weights of the neurons that are a neighbor of the winning neuron. The weight values of the input nodes are then normalized again. The

weights will eventually spread out and the neighborhoods will become sensitive to certain input patterns.

Kohonen's SOM has been successfully applied in the areas of speech recognition, image processing, robotics, and process control [Ref. 4].

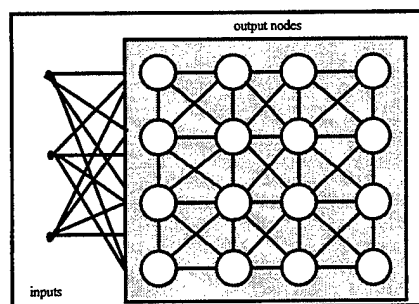


Figure 2.7: Kohonen self organizing map [Ref. 8].

Adaptive resonance theory models (ART) are able to continually accept new patterns until some threshold is met, namely the number of output nodes. ART models use the competitive learning rule so they are fully connected, both forward and backward, networks that start out with none of the output neurons producing any pattern. As inputs are fed to the network, output nodes are assigned to the new input. The input is first compared against the stored patterns and the closest match is selected for further comparison against a vigilance. A vigilance is a predetermined value between 0 and 1. The closer to one the closer the match must be, the closer to zero the weaker the match is allowed to be. The two patterns are compared by computing the dot product of the patterns and dividing by the number of ones in the input pattern. If the ratio is greater than the vigilance then the pattern is considered similar and the stored pattern is updated. If the ratio is not greater than the vigilance and there is a free output node, it is assigned to the new pattern. If there is no free output node then nothing is changed.

One of the main problems with this model is that noisy input could create more than one node that represents the same input. ART models are used for categorization problems.

D. SUMMARY

The clearing of military bases is an inherently dangerous and costly mission. The traditional method of digging up every piece of metal that returns a signal from the metal detector needs to be replaced with a more efficient method. Artificial neural networks possibly present a cost efficient method of dealing with UXO's. At the center of all neural networks is the model of the biological neuron. There are many types of neural network architectures that use various types of learning rules to solve a variety of problems. Networks use supervised (use a teacher), unsupervised (learn on there own) or reinforcement learning (a hybrid of the two). They are either associative (have memory) or non-associative (no memory). Neural networks are used to solve problems such as pattern classification, categorization, function approximation, prediction, and data analysis. In range remediation there are three problems: detection, localization, and classification. The problem of determining what type of ammunition is on the ground is a classification problem. A multi-layered (three layer) feed-forward neural network with supervised learning (back-propagation) for pattern classification was chosen for this thesis. The next chapter will present the process by which the neural network design for UXO detection was chosen as well as the details of the network design.

III. ARTIFICIAL NEURAL NETWORK DESIGN

A. CHOOSING A NETWORK DESIGN

Once the decision has been made to use a neural network to solve a problem, the next step becomes choosing an architecture to fit the needs of the problem. There are several types of neural networks for every type of problem. Table 3.1 shows a few of the types of neural networks and what types of problems they solve. Paradigm, learning rule, architecture, and learning algorithm were all discussed to some extent in Chapter II. The goal of this thesis is to determine the type of ammunition presented to a neural network as recorded from a magnetometer. This is a classification problem, so the focus will be on what type of classification neural network will best suit the needs of the UXO project. There are several factors to take into account.

1. Design Factors

a. Speed of Execution

As in all real time systems, speed is at the top of the priority list. Speed of execution refers to the time it takes a neural network to determine what kind of object the input data represents. This is not to be confused with the rate at which a neural network learns. Learning rate is the time it takes a network to be trained to recognize a set of inputs. Learning rate will be discussed later. The more connections there are the slower the network will operate. All of the associative memory networks require connections to previous nodes in order to store the information on a pattern. This will slow the performance of the network. A strictly feed-forward network will limit the number of connections needed to accomplish the mission. In order for this project to perform correctly, the neural network more than likely will be implemented in hardware. A hardware implementation can speed up the execution speed by as much as 1000 times.

b. Generalization

Generalization refers to the type of input. If the input is the same as the input that the network was trained on, then generalization is not that important. But, if the input is something the network has not seen before, then the ability of the network to generalize will make or break the system. In order to ensure the former case, the network would have to be trained on an exhaustive list of possibilities. This is usually not possible. The latter case may lead to the network failing when an unknown is introduced. If your network is designed to generalize, it may try to give an answer for something it really should not recognize. As long as it is a low percent of probability there should not be a problem.

c. Scalability

Some systems are able to add new patterns without any rewriting of code. But, there is a finite number of patterns that can be added to any network. The more patterns that are programmed in to a network, the more connections there are and the slower the network will run. A designer must determine if the set of possibilities are relatively small or infinite in size.

d. Learning Speed

Learning speed are a major area of concern in any neural network architecture. It can take several iterations for a training algorithm to converge on the optimal solution. Sometimes it may even take several restarts to get to the optimal solution without becoming stuck in a local minimum. A network that takes a longer time to train will more than likely come up with a better optimal solution than a fast training algorithm. As long as training time does not interfere with the execution time, learning speed should be a matter of quality not quantity. Chapter II covered some of the types of algorithms that determine the speed with which a network will learn. Different architectures use different training algorithms.

e. Number of Layers

This is a topic of great interest for any designer of a network. What is the magic number of layers and neurons per layer? Trial and error seems to be the best solution. There are however, a few guidelines for determining these parameters. The number of layers in a network seems to vary according to the problem at hand. Figure 3.1 shows the ability of different numbers of layers to solve different levels of complex problems. When dealing with the number of neurons in a layer, one method is having more than three times as many nodes in the second layer as in the first layer [Ref. 8].

Figure 3.1 gives the definition of a three layer network as one with three layers of neurons. Some books present the two layer network from Figure 3.1 as a three layer network because the input values are counted as a layer. While the input layer does not perform any computation, it is still considered a layer. This thesis refers to the two layer network in Figure 3.1 as a three layer network. In general there is a clear advantage to using a single hidden layer of non-linear neurons between the input and output layers, but having more than two hidden layers in a system with non-linearity does not increase their computational power [Ref. 9]. It is said that a three layer network can solve any problem [Ref. 10].






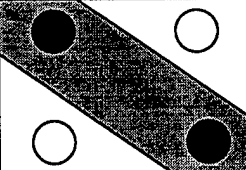
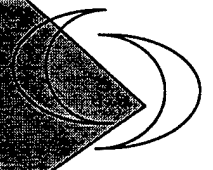
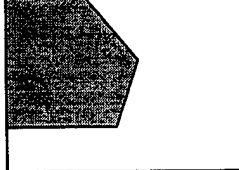

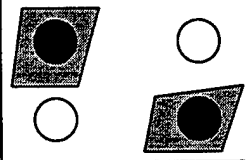
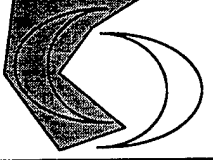
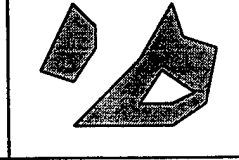
Structure	Description of decision regions	Exclusive -OR problem	Classes with meshed regions	General region shapes
 Single layer	Half plane bounded by hyperplane			
 Two layer	Arbitrary (complexity limited by number of hidden units)			
 Three layer	Arbitrary (complexity limited by number of hidden units)			

Figure 3.1: A geometric interpretation of the role of hidden units in a two-dimensional space [Ref. 4]

f. Connectivity

A fully connected network produces a lot of overhead and may cause the system to slow down. Networks that are not fully connected are feature extraction networks. Feature extraction may be necessary with a large input set. The number of connections and time to produce an answer, may be so great with a fully connected system that feature extraction may be necessary. Defining features and how they tie in can be a complex problem. A fully connected model is easier to follow.

<i>Paradigm</i>	<i>Learning rule</i>	<i>Architecture</i>	<i>Learning Algorithm</i>	<i>Task</i>
Supervised	Error-correction	Single or multi-layer perceptrons	Perceptron learning algorithms	Pattern classification
			Back-propagation	Function approximation
			Adaline & Madaline	Prediction, control
	Boltzmann	Recurrent	Boltzmann learning algorithm	Pattern classification
	Hebbian	Multi-layer feed-forward	Linear discriminant	Data analysis
	Competitive	Competitive	Learning vector quantization	Pattern classification
				Within-class categorization
		ART network	ART Map	Data compression
				Pattern Classification
Unsupervised	Error-correction	Multi-layer feed-forward		Within-class categorization
	Hebbian	Feed-Forward or competitive	Sammon's project	Data analysis
	Competitive	Competitive	Principal component analysis	Data analysis
				Data compression
			Vector quantization	Categorization
				Data compression
		Kohonen's SOM	Kohonen's SOM	Categorization
Hybrid	Error-correction and competitive	RBF network		Data analysis
			ART1, ART2	Categorization
			RBF learning algorithm	Categorization
				Pattern classification
				Functional approximation
				Prediction, control

Table 3.1: A layout of different types of architectures and the task they perform [Ref. 4].

2. Choosing an Architecture

Of the classifiers, perceptron, multi-layer feed-forward, Hopfield, SOFM, ART, Boltzmann, and Hebbian, the multi-layer feed-forward neural network provides the best architecture for the UXO problem. The perceptron is limited in the number of decision regions it is able to separate to one. Boltzmann and Hopfield rely on binary input and are constantly training. Continuous data will be used for the UXO problem. SOFM, ART, and Hebbian networks rely on associative memory which will slow the network down when considering the number of nodes that are needed for this problem.

The multi-layered feed-forward network has a fast execution time because training has already been completed and it is not an associative memory network. This means the network only has to feed-forward while making a decision and not backward as well. A fully connected multi-layered network will produce more overhead than a feature extraction network, but still less than an associative memory network. This type of network is able to generalize quite well. The multiple number of layers allows the network to separate complex decision regions as shown in Figure 3.1. Once the network is established, one must create a new network in order to expand the output set or combine two networks. Both methods will require retraining. The back-propagation algorithm is slow in convergence, but once the network is trained, the algorithm is removed.

B. MULTI-LAYERED FEED-FORWARD NETWORK

The multi-layered feed-forward neural network has been chosen as the architecture for this thesis. It can be shown that a three layer network consisting of an input layer, one hidden layer, and an output layer, can represent any function provided there are a sufficient number of neurons in the hidden layer [Ref. 10]. For this reason, without any other information on how to start a neural network, the three layer network is kind of the standard. Therefore, a three layer network has been chosen in order to cut down on the total number of neurons and for its simplicity.

The input layer of the network is fully connected to the hidden layer. This means that for each neuron in the hidden layer, there is a weight that is associated with a corresponding neuron in the input layer. The input layer in this network serves as only an input node. The values from each input neuron are passed on to all of the neurons in the hidden layer. Figure 3.2 illustrates how the values are passed to the hidden layer.

The hidden layer is where the first computations take place. Each neuron in the hidden layer takes the weighted sum of all of the values from the input layer. Each neuron then runs the weighted sum through a non-linear activation function in order to get the output value for each neuron in the hidden layer. The activation function to be used is the logistics function which will be discussed more in the next section. Now the process starts all over again. This time the input is the output from the hidden layer.

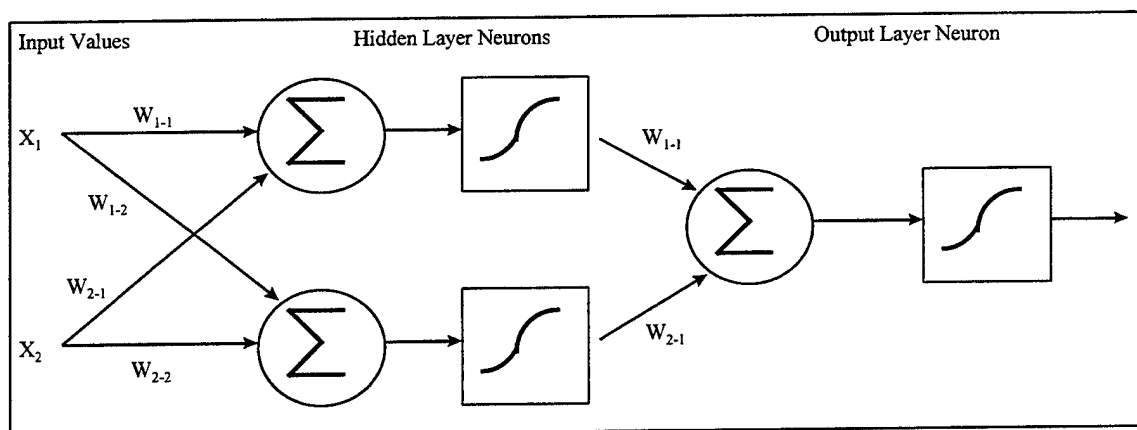


Figure 3.2: Multi-layered feed-forward neural network.

The output layer receives the output values of the hidden layer as its input values. The number of nodes in the output layer corresponds to the number of different elements in the training set. For example, if your net is set up to classify 5 different items, then there are five neurons in the output layer. Referring to Figure 3.2, the same process starts all over again. The weighted summation of the input values multiplied by the weights are fed through the activation function. The highest value of the output layer neurons is the answer for the particular input data fed to the network. Notice that each neuron in the output layer has the same number of weights associated with it as the number of neurons

in the hidden layer. How these weights are derived is the topic of the next section. Figure 3.3 shows how the input values are fed forward through a feed-forward network.

The following is an outline of the feed-forward process in a three layer neural network:

- Calculate the weighted sum for the hidden layer. The weighted sum simply takes the input values of all of the nodes connected to a neuron and multiplies those values by their associated weights. The resulting values are then summed.

$$weighted_sum_i = \sum_{k=1}^j input_value_k * weight_{ki}$$

Where i is the neuron in the hidden layer and k is the input neuron.

$$H_1 = (.98 * 1) + (.65 * 2) + (.50 * 1) = 2.78$$

$$H_2 = (.98 * 2) + (.65 * 2) + (.50 * 3) = 4.76$$

$$H_3 = (.98 * 1) + (.65 * 1) + (.50 * 1) = 2.13$$

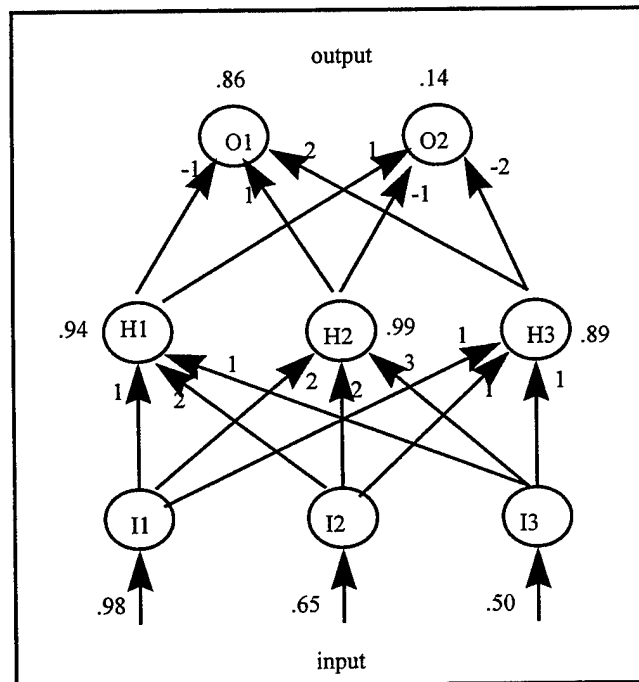


Figure 3.3: Feed-forward network

- Calculate the output value. The output value for the neuron is the weighted sum fed through an activation function. The logistic function will be used in this network.

$$output_value_i = logistic_function(weighted_sum_i)$$

Where i is the neuron in the hidden layer.

$$H_1 = 1 / (1 + e^{-2.78}) = .94$$

$$H_2 = 1 / (1 + e^{-4.76}) = .99$$

$$H_3 = 1 / (1 + e^{-2.13}) = .89$$

- Calculate the weighted sum for the output layer. This is the same as for the hidden layer. The input value is know the output value of the hidden layer

$$weighted_sum_i = \sum_{k=1}^j input_value_k * weight_{ki}$$

Where i is the neuron in the output layer and k is the hidden layer neuron.

$$O_1 = (.94 * -1) + (.99 * 1) + (.89 * 2) = -1.83$$

$$O_2 = (.94 * 1) + (.99 * -1) + (.89 * -2) = 1.83$$

- Calculate the output value. Use the same process as the hidden layer.

$$output_value_i = logistic_function(weighted_sum_i)$$

Where i is the neuron in the hidden layer.

$$O_1 = 1 / (1 + e^{-1.83}) = .86$$

$$O_2 = 1 / (1 + e^{1.83}) = .14$$

The output values .86 and .14 are the actual outputs for the network. In this case the desired output was 1 and 0. The next step is to adjust the weights in order to reach the desired output. This process is accomplished through the back-propagation algorithm.

C. BACK-PROPAGATION ALGORITHM

Training an artificial neural network deals with optimizing the weights associated with each individual neuron in order to produce the desired results. In a truly feed-forward network, each neuron contributes to the overall error produced by the output neurons. The error is produced by having the wrong weight values. It is impossible to guess the correct value of each weight in the network. A method must be established to choose these weight as to produce the correct output. The back-propagation algorithm is one way of solving this problem.

The development of the back-propagation algorithm has made the multi-layer neural network the most popular of the artificial neural networks. The back-propagation algorithm seems to have been developed simultaneously by Rumelhart, Hinton, and Williams in 1986, Parker in 1986, and LeCun in 1985 [Ref. 7]. The algorithm is rather straight forward. If a designer has no idea what the weights for his network should be, which is almost always the case, he can start with random weights and let the back-propagation algorithm determine what the values should be. An optimal value for all of the weights is hard to achieve, but determining the values within an acceptable level of error is usually achievable. Based on an acceptable level of error, the back-propagation algorithm adjust the weights until the level of error is within a given tolerance. Zero tolerance on error is an unrealistic goal, rather a small percentage of acceptable error is used.

The whole key to the algorithm is based on the gradient descent problem illustrated in Figure 3.4. Adjusting weights must be done a little at a time in order to get the best possible answer. The step toward the optimal solution is called the learning rate. If too large of steps are taken, the training algorithm may jump back and forth on the bowl and never reach an optimal solution. A very small step down the side of the bowl will translate into a significant change in the x value. Figure 3.4 illustrates the change in the value of x and how it will effect the ability of the net to reach the global minimum, also known as the optimal solution. There are also local minimums that can be reached if

the step is too small. This means that the “optimal” values for the weights will not produce the optimal solution. The way to get around falling into this trap is to choose a small step in exchange for a long training period, or use trial and error by retraining the network until you are satisfied with the results. Either way is time consuming.

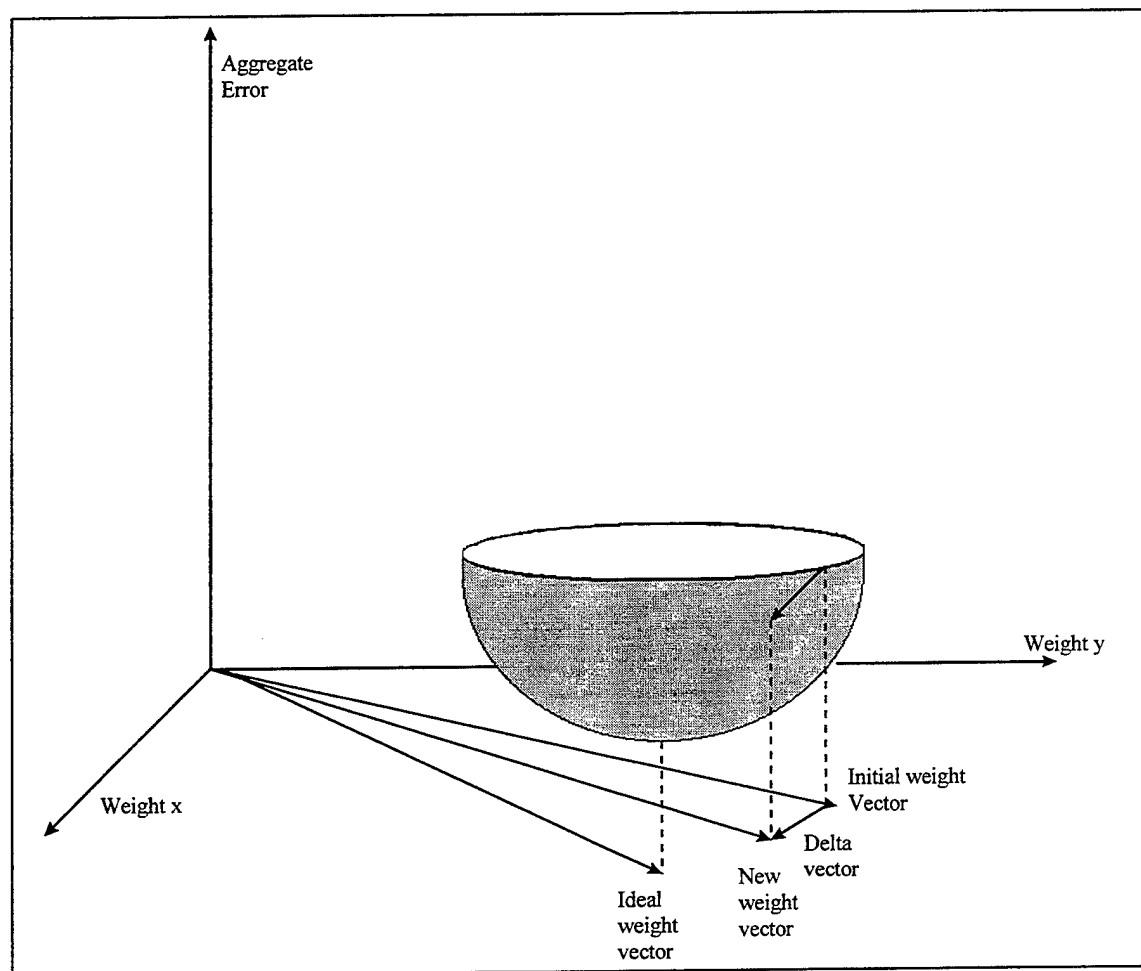


Figure 3.4: A pictorial representation of the gradient descent problem.

Another method often employed when training a network is to use a momentum variable. The idea behind the momentum variable is to take big steps with the gain variable and then the momentum variable is applied in order to further fine-tune the network towards the optimal value for the weights. This method creates the effect of speeding towards the optimal solution and then putting on the brakes and gradually

slowing down. The whole idea is to stop as close to the optimal solution without going beyond the optimal solution and heading back up the other side of the bowl in Figure 3.4.

The multi-layered feed-forward neural network shown in Figure 3.3 will be used to demonstrate how the back-propagation algorithm was employed in this thesis. The back-propagation algorithm is outlined as follows using a three layer feed-forward network:

- Calculate the output error for each neuron in the output layer. After the inputs have been fed through the network and the output layer produces an answer for each neuron in the output layer, the answer is compared to the expected output answer.

$$\text{Output Error} = \text{Expected} - \text{Actual}$$

$$\text{Output Error } O_1 = 1 - .86 = .14$$

$$\text{Output Error } O_2 = 0 - .14 = -.14$$

- Calculate the total error for each element in the training set. The total output error is calculated by summing the absolute values of each node in the output layer. This is the total error for that element in the training set.

$$\text{TotalOutputError} = \sum_{j=1}^i |\text{OutErr}_j|$$

Where i is the number of neurons in the output layer.

$$\text{Total Output Error} = |.14| + |-.14|$$

- Calculate the total error for the training set. This variable will be updated as every element in the training set is feed through the network.

$$\text{Total Error} = \text{Total Error} + \text{Total Output Error}$$

- Calculate the incoming error for the middle layer. The amount of error the middle layer contributes to the output error is a two step process. First one must notice that each node in the middle layer contributes an error to all of the nodes in the output layer. With this in mind, the error at each output node must be multiplied by the weight associated with the node in the middle layer.

These values are then summed to produce the incoming error for that particular node in the middle layer.

$$IncomErr = \sum_{j,k=1}^i (OutputError_j * weight_k)$$

Where j and k are which output node.

$$\text{Incoming Error } H_1 = ((.14 * -1) + (-.14 * 1)) = -.28$$

$$\text{Incoming Error } H_2 = ((.14 * 1) + (-.14 * -1)) = .28$$

$$\text{Incoming Error } H_3 = ((.14 * 2) + (-.14 * -2)) = -.56$$

This is the method used for dealing with propagating the output layer error back in this thesis. Another method used is to figure the derivative of the output layer error in order to propagate the actual error back. The above method used in this thesis allows for a faster training period.

- Calculate final error for each middle layer node. Calculating the final error takes into account that an activation function was used to produce the error value being propagated back. Therefore the values from the calculate incoming error step must be fed back through the activation function. This is achieved by calculating the derivative of the activation function. This is why a non-linear function is used as the activation function. An activation function with a smooth non-linear transition works the best. The logistic function is used due to the ease of calculating the derivative.

$$Final_Error = Incomming_Error * Derivative_of_activation_function$$

For the logistics function, the derivative is the value at the middle layer node multiplied by one minus the value at the middle layer node. See Figure 3.5.

$$\text{Final Error } H_1 = -.28 * .94(1 - .94) = -.0158$$

$$\text{Final Error } H_2 = .28 * .99(1 - .99) = .0028$$

$$\text{Final Error } H_3 = .56 * .89(1 - .89) = .0548$$

- Calculate weight change for each output layer weight. At this stage in the algorithm the learning rate must be taken into account. Since the amount of

error is now known, how far to move towards the optimal solution without stepping over that solution must be determined. That is the function of the learning rate. A large learning rate may miss the optimal solution, however a small learning rate will take longer to train.

$$\Delta Weight_j = \beta * OutputError_i * InputValues_k$$

Where j is the weight to be adjusted, i is the error at the output node, k is the value from the middle layer node and β is the learning rate.

$$\Delta O_1 W_1 = .2 * .14 * .94 = .0263$$

$$\Delta O_1 W_2 = .2 * .14 * .99 = .0277$$

$$\Delta O_1 W_3 = .2 * .14 * .89 = .0249$$

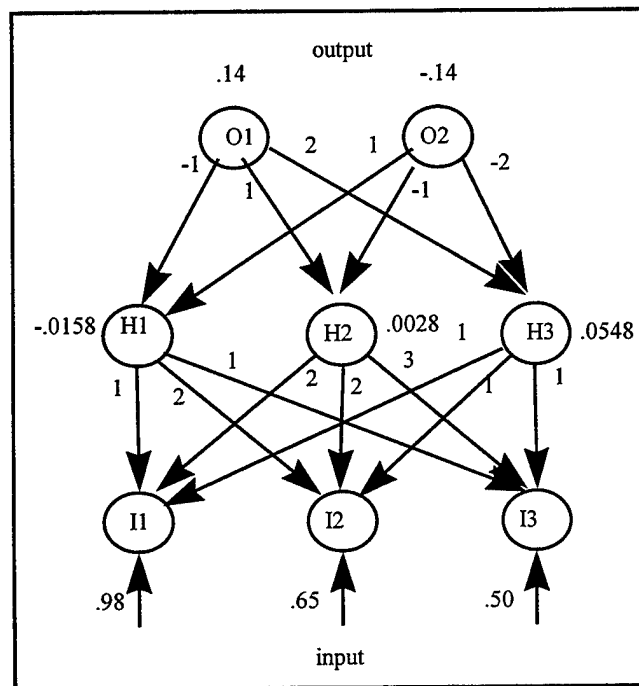


Figure 3.5: Back-propagation

- Change weights for each output layer weight. Here all that is need is to add the weight change calculated in the above step to the current weight value.

$$W_1 = -1 + .0263 = -.9737$$

$$W_2 = 1 + .0277 = 1.0277$$

$$W_3 = 2 + .0249 = 2.0249$$

- Calculate weight change for middle layer weights. This step follows the same equations as the output layer.

$$\Delta Weight_j = \beta * OutputError_i * InputValues_k$$

$$\Delta H_1 W_1 = .2 * -.0158 * .98 = -.0031$$

$$\Delta H_1 W_2 = .2 * -.0158 * .65 = -.0021$$

$$\Delta H_1 W_3 = .2 * -.0158 * .50 = -.0016$$

- Change weights for each middle layer weight.

$$W_1 = 1 + -.0031 = .9969$$

$$W_2 = 2 + -.0021 = 1.9979$$

$$W_3 = 2 + -.0016 = 1.9984$$

- Repeat process for each element in the training set. Every element in the training set must be fed through and the weights adjusted once for each element. Then in the next step, the sum of the errors for each element in the training set is compared against the acceptable level of error. If the error is not in tolerance, the whole process is repeated. This prevents the training of each element in the training set to the acceptable level of error one at a time. The results of that method would be adjusting the weights for the first element then the next and finally the last one. However this would lead to the net being training only for the last element in the training set and not the other elements. Figure 3.6 shows the results of the above calculations for one side of the network.

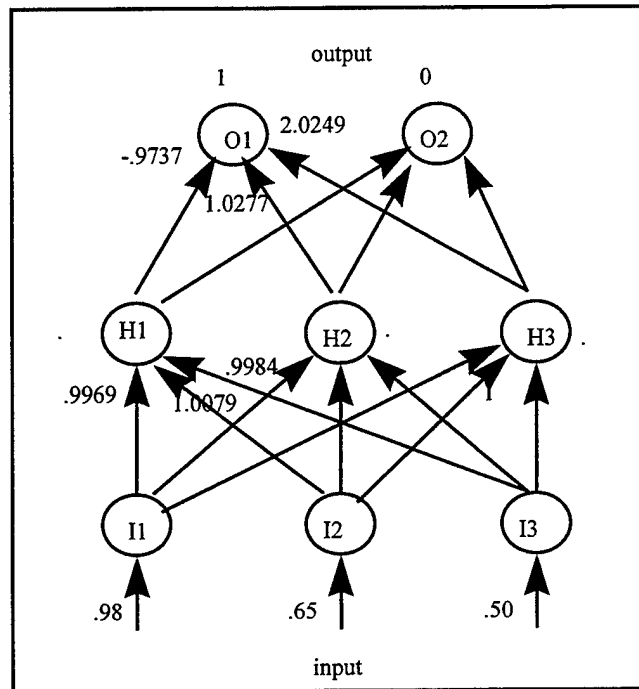


Figure 3.6: Adjusted weights

- Repeat the entire process until Total Error is at an acceptable level. This thesis uses the method of totaling up the errors and comparing them to an acceptable level of error. Other methods include the least mean square and standard deviation. Whatever the method, the goal is to determine the stopping point of the training process.

That was just one iteration of the back-propagation algorithm. Usually many iterations are needed in order to reach the acceptable level of error imposed by the designer. Choosing a large learning rate can also lead to jumping back and forth across the bowl as described in Figure 3.4. If the network did not get stuck in a local minimum, it now contains an acceptable solution for the given training data that is within the chosen acceptable level of error. Training a network is usually very computationally expensive. The actual ammunition neural network is described in the next chapter.

IV. AMMUNITION NETWORK

A. INTRODUCTION

The intent of this chapter is to explain in detail the entire process of setting up the ammunition network. There are a significant number of tasks that need to take place before an artificial neural network can be implemented. First, the training set must be chosen. The training set is the set of objects that the neural network is to identify. Data preparation is probably the most important phase. The accuracy of the network is based upon the training data that is presented to it. So much relies on the data that it is important to recognize if this part fails the whole experiment fails. And last, but not least, the network must be implemented.

B. AMMUNITION TYPES

The ammunition used for this thesis was chosen based upon commonly found types in impact areas and availability. A U.S. Army ordnance officer, CPT Paul Arcangeli, was consulted for the training set and the items were chosen based on their availability at a local ordnance unit. The training set was limited to five common rounds due to the amount of time necessary to record the data, and that this should demonstrate a proof of concept. The following is a list of the types of ammunition used.

- 60mm mortar round (Figure 4.1)
- 81mm mortar round (Figure 4.2)
- 105mm artillery round (Figure 4.3)
- 105mm high explosive anti-tank round (Figure 4.4)
- 3.5 inch rocket (Figure 4.5)

The training set is made up of two distinct sets of the above ammunition and is displayed in Figures 4.6 and 4.7. The test set was derived from separate readings of the second set of ammunition.

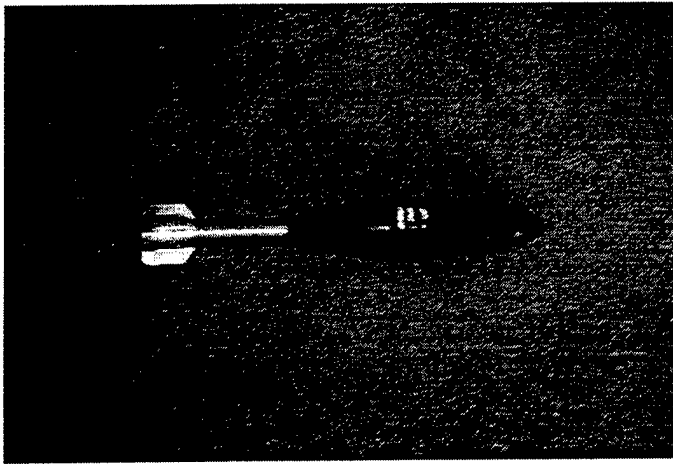


Figure 4.1: 60mm mortar.

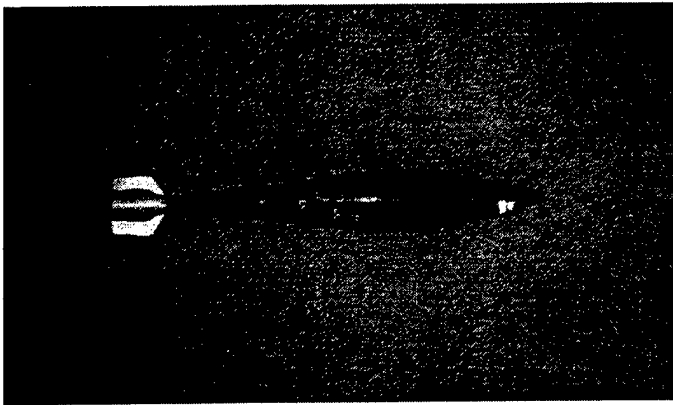


Figure 4.2: 81mm mortar.

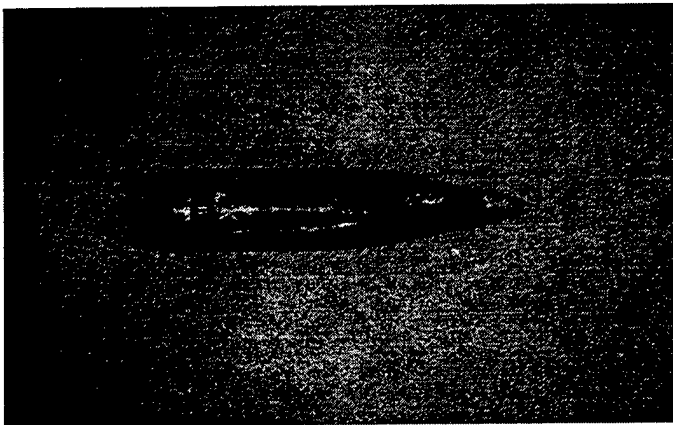


Figure 4.3: 105mm artillery round.

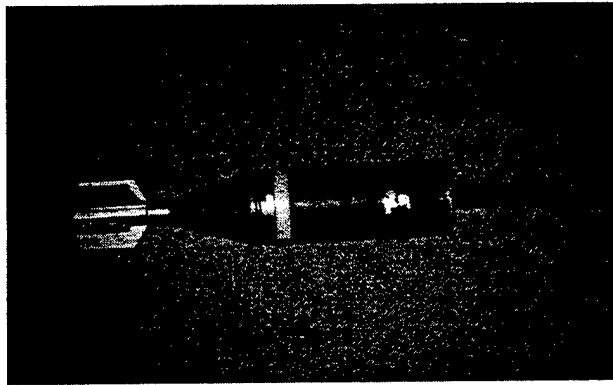


Figure 4.4: 105mm HEAT round.

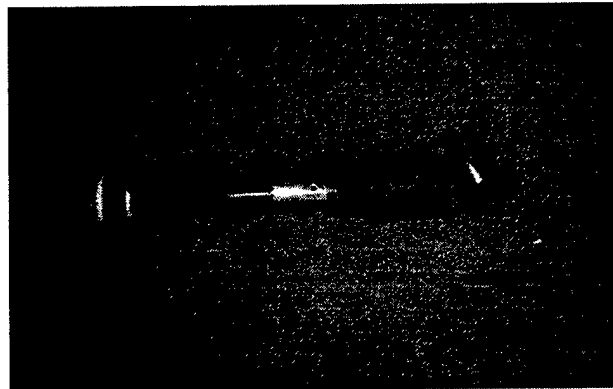


Figure 4.5: 3-5in rocket.

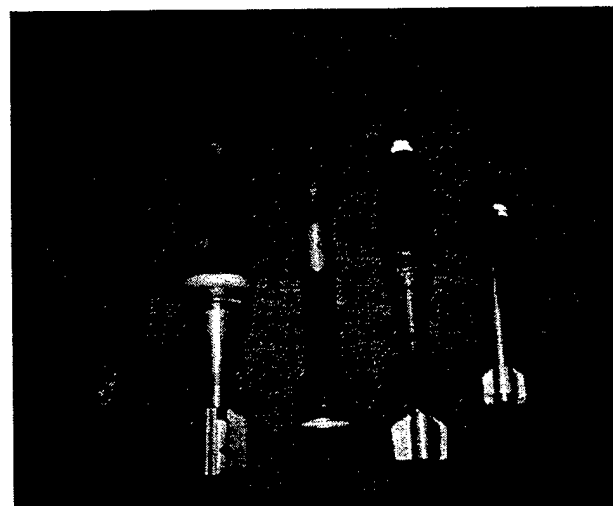


Figure 4.6: Training set 1.

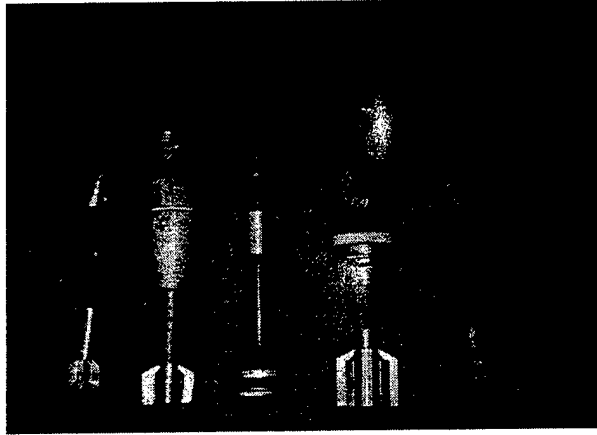


Figure 4.7: Training set 2.

C. DATA PREPARATION

1. Magnetometer

As was stated in Chapter I, the magnetometer is the sensor that was used in this thesis. The GA-72Cd Magnetic Locator has an analog audio signal and a digital display of the signal. The display shows the strength of the signal and the polarity of the signal. The magnetometer measures the difference in signal strength between two sensors located in the shaft of the magnetometer. One sensor is placed towards the bottom of the shaft and the other is towards the top of the shaft. The digital readout of the signal is a three digit number ranging from -36.0 to 36.0. The orientation of a piece of metal might be determined by the polarity of the signal. The polarity is indicated by the positive or negative reading. A 175mm projectile can be detected up to five feet in the ground. There are four sensitivity levels on the magnetometer [Ref. 2]. Sensitivity level 2 was used for the data collection. The sensitivity level ranges from 1 to 4, with level 4 being the most sensitive.

Placement of the magnetometer in reference to the round played an important roll. The strength of the signal varies with the distance from the bottom of the magnetometer to the round. For this reason, the distance from the magnetometer to the round was varied in the two training sets. As will be discussed in the conclusions, this fact will lead to a

very large training set when depths and orientations are taken into account. In order to limit the size of future training sets some assumptions must be made. The orientation of the round will need to be determined based on the centroid of the round and the polarity at each end. If this can be determined then the data can be oriented based off a known orientation. It is hoped that there is an automatic technique that will bring the data of a buried round to the reference point of surface laid. After these two problems are solved the next step is to account for possible angles of the round.

2. Data Collection

The readings were taken at the Naval Postgraduate School Beach in Monterey, California. Therefore, the terrain was sand. The zero reading for the magnetometer was a +.08. A 31cm x 35cm grid was used to record the data. Each grid square was 2cm x 2cm. This grid square setup produced 1085 readings per piece of ammunition. The data was recorded on site and entered into an input file for later use with the neural network.

An important part of the data collection was a wooden frame on which the magnetometer was mounted (Figure 4.8). The frame allow for the magnetometer to be moved with two degrees of freedom along the x and y axis of the grid square. The placement of the round in the frame was very important for the neural network to be able to recognize a round it had seen before. The following list will specify the positions of each round.

- 60mm mortar - Top - 35.5cm from left edge of grid, 12.5cm from top edge of grid . Bottom - 35.5cm from left and 20 cm from bottom. Highest point on round - 4 cm from bottom of magnetometer.
- 81mm mortar - Top - 35.5cm from left, 7cm from top. Bottom - 35.5cm from left, 5cm from bottom. Highest point - 2.5cm.
- 105mm heat round - Top - 36.5 cm from left, 5cm from top. Bottom - 36.5cm from left, 4cm from bottom. Highest point - 2.5cm.

- 105mm artillery round - Top 37cm from left, 14.5cm from top. Bottom - 36cm from left, 12.1cm from bottom. Highest point - 2.5cm.
- 3-5in rocket - Top - 36.5cm from left, 5cm from top. Bottom - 36.5cm from left, 12cm from bottom. Highest point - 2.5cm.

The above measurements apply to the second training set and the test set. The positions of the first training set were centered in the frame. This will allow for a variation in the overall training set and allow the network a different look at the same type of round. The readings are different for the two training sets due to the precision of the placement. As will be discussed in the conclusions, this gives an indication of the number of possibilities of training sets.

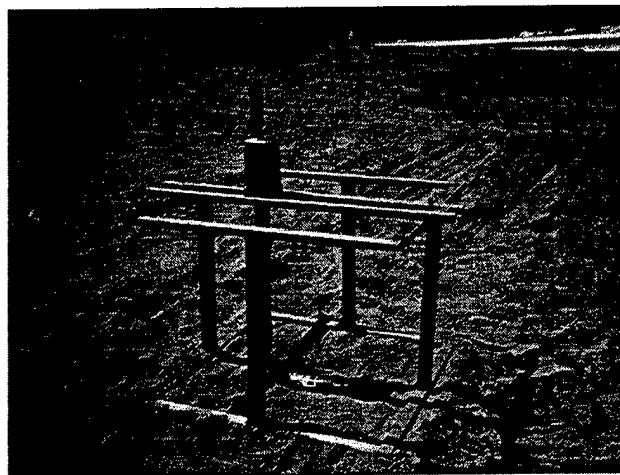


Figure 4.8: Wooden frame and magnetometer.

D. LISP IMPLEMENTATION OF NETWORK

Lisp was chosen as the prototyping language for the neural network due to the ease of list manipulation of each input and weight pairs. If the input and the weights are viewed as a list, Lisp will allow for easy manipulation. An object oriented approach was taken to the implementation of the multi-layered feed-forward neural network. Having no idea of the number of layers or number of neurons in each layer that is needed to get

the best solution, the object oriented approach allows for new instantiations of neurons and layers if the design were to change. Unfortunately, with a relatively large input size, this could lead to a substantial overhead. The neural network is made up of neuron objects, layer objects and a network object.

Objects in Lisp are instantiations of classes. Classes contain slot values which store the values that belong to the object. Slot values are the equivalent to data members in C++ . Methods are the functions that are allowed to interact with the slot values. They are equivalent to member functions in C++. The following classes will be explained in terms of slot values and methods. The *back-propagation.lisp* file is in Appendix D and contains the algorithm for training the network. Appendix E contains the user interface functions for creating the objects, building the input data and saving the weights associated with the network. For further explanation of the code, see the appendices. The comments should serve as a guide as to what is happening.

1. Neuron Class

The neuron object contains the slot values *output-value* and a *weight-value*. The code is in Appendix A. Methods include the *initialization-neuron* and *neuron-activation*. The slot values *weight-value* and *output-value* are accessed by there respective accessor names *weight-vector* and *output-vector*.

```
(defclass neuron ()
  ((weight-value :accessor weight-vector
                 :initarg :weight-value
                 :initform '())
   (output-value :accessor output-vector
                  :initarg :output-value
                  :initform '(1))))
```

The *initialize-neuron* method takes as parameters, a neuron object, an *input-length*, and a *weight-vector*. If a weight list is not passed to the *initialization* method, random weights are created via *make-random-weights* function. In order to determine how many random weights to create, *make-random-weights* is passed *input-length*. A

random variable seed is also passed to *make-random-weights*. If a weight list is sent to *initialize-neuron* then *weight-vector* is set to that list.

```
(defmethod initialize-neuron ((my-neuron neuron)
  input-length weight-vector1)
  (if (null weight-vector1)
      (setf(weight-vector my-neuron)
            (make-random-weights input-length 2.0))
      (setf(weight-vector my-neuron) weight-vector1)))
```

The *neuron-activation* method calls *my-summation* which returns the summation of the input data multiplied by the weight vector for that neuron. This value is fed to the activation function by the call to the *sigmoid* function. The value returned from the *sigmoid* function is stored in the *output-value* slot of the neuron.

```
(defmethod neuron-activation ((my-neuron neuron) layer-input)
  (setf (output-vector my-neuron)
        (list (sigmoid (my-summation (weight-vector my-neuron)
                                     layer-input))))))
```

2. Layer Class

The layer object is a subclass of the neuron class. The code is in Appendix B. It inherits all of the properties of the neuron class plus a list of the neurons that are in that layer, called *node-value*, an *input-value* slot containing the data input values, and a *number* slot corresponding to the number of neurons in the list. *Node-list*, *input-vector*, and *number-value* are the accessor names for *node-values*, *input-value*, and *number* respectively.

```
(defclass layer (neuron)
  ((node-value :accessor node-list
               :initarg :node-value
               :initform'((make-instance 'neuron)))
   (input-value :accessor input-vector
                :initarg :input-value
                :initform '())
   (number :accessor number-value
           :initarg :number
           :initform 1)))
```

Layer class has four methods: *initialize-layer*, *initialize-node-list*, *activate-layer* and *activate-layer-list*. *Initialize-layer* sets all of the slot values of layer class. *Initialize-layer*'s parameters are a layer object and the layer parameters. The layer parameters contain the number of neurons in the layer, the input list, and the weight list. The *node-list* is set by calling *build-layer* which creates the neurons in the *node-list*. *Initialize-layer* then sets the values of the neurons by calling *initialize-node-list*.

```
(defmethod initialize-layer ((my-layer layer) layer-params)
  (setf(number-value my-layer) (first layer-params))
  (setf(input-vector my-layer) (second layer-params))
  (setf(weight-vector my-layer) (third layer-params))
  (setf(node-list my-layer) (build-layer (first layer-params)))
  (initialize-node-list my-layer layer-params))
```

Initialize-node-list iteratively steps through the *node-list* and taking one neuron at a time and calls *initialize-neuron* which sets all of the slot values of the neuron. Notice that the weight list for the neuron must also be sent to *initialize-neuron*. If the weight list is null, the call to *initialize-neuron* contains the null weight list, else the weight list is stepped through in the same manner as the *node-list*.

```
(defmethod initialize-node-list ((my-layer layer) layer-params)
  (do* ((i 0 (+ i 1))
        (neuron1 (first (node-list my-layer))
                  (nth i (node-list my-layer)))
        (weight1 (first (third layer-params))
                  (if (null (third layer-params))
                      (first (third layer-params))
                      (nth i (third layer-params)))))
        ((> i (- (length (node-list my-layer)) 1)))
    (initialize-neuron neuron1
                      (length (input-vector my-layer)) weight1)))
```

The method *activate-layer* sets the *output-vector* slot of the layer to the list of output values returned by the call to the method *activate-layer-list*. *Activate-layer-list* iteratively steps through every neuron in the *node-list* and calls *activate-neuron*. The results of *activate-neuron* are consed together to build the *output-vector*.

```

(defmethod activate-layer-list ((my-layer layer))
  (do* ((i 0 (+ i 1))
        (layer1 (cons (first (neuron-activation
                              (first (node-list my-layer)) (input-vector my-layer))) ())
              (cons (first (neuron-activation (nth i (node-list my-layer))
                                                (input-vector my-layer))) layer1)))
    )
    ((> i (- (length (node-list my-layer)) 2)) (reverse layer1))
  ))

```

3. Network Class

The network class is a subclass of the layer class. The code is in Appendix C. It inherits all of the slot values of the layer class. Since the layer class is a subclass of the neuron class, the network class also inherits all of the slot values of the neuron class. The only additional slot value is the *nodes-layer-value* which can be accessed by the *nodes-per-layer* accessor.

```

(defclass network (layer)
  ((nodes-layer-value :accessor nodes-per-layer
                      :initarg :nodes-layer-value
                      :initform '(1))))

```

The network class contains the four methods: *initialize-network*, *initialize-network-layers*, *activate*, and *activate-network*. They all serve the same purposes as the methods in the layer class just at a higher level of abstraction. *Initialize-network* sets all of the slot values to the values of the network parameters that are passed in. The network parameters consist of a list that contains two list. The first list is the number of neurons, the input data, and the weight list for the first layer. The second list is the number of neurons, input data and weight list for the second layer. The input data for the second layer is initialized to the input data for the first layer, but it is changed to the *output-vector* of the first layer when the network is activated. It also calls *build-network* which makes instances of layers.

```
(defmethod initialize-network ((my-network network) network-params)
  (setf(input-vector my-network) (second (first network-params)))
  (setf(number-value my-network) (length network-params))
  (setf(nodes-per-layer my-network) (list(first(first network-params))
                                           (first(second network-params))))
  (setf(node-list my-network) (build-network
                               (length network-params)))
  (initialize-network-layers my-network network-params) )
```

Initialize-network-layers strips off each layer from the *node-list* of the network and passes it to *initialize-layer* which sets up the layer and calls *initialize-neuron* to initialize each neuron in the layer.

```
(defmethod initialize-network-layers((my-network network)network-params)
  (do* ((i 0 (+ i 1))
        (layer1 (first (node-list my-network)) (nth i (node-list
                                                         my-network)))
        )
    ((> i (- (length (node-list my-network)) 1)))
    (initialize-layer layer1 (nth i network-params)))
  )
```

Activate sets the *output-vector* of the network to the value returned from *activate-network*. *Activate-network* strips off one layer at a time and passes it to *activate-layer* which sends each neuron in the layer to *activate-neuron*. It also sets the input value of the second layer to the output value of the first layer.

```
(defmethod activate ((my-network network) my-network-input)
  (set-network-input my-network my-network-input)
  (activate-network my-network)
  (setf(output-vector my-network)
        (output-vector (first (last (node-list my-network))))) )

(defmethod activate-network ((my-network network))
  (do* ((i 0 (+ i 1))
        (output-layer(activate-layer (first (node-list my-network)))
        (if (nth i (node-list my-network))
            (activate-layer (nth i (node-list my-network)))))
        )
    ((> i (- (length (node-list my-network)) 1)))
    (if (nth (+ i 1) (node-list my-network))
        (setf(input-vector (nth (+ i 1) (node-list my-network))
                    output-layer))))
```

4. Back-propagation Algorithm

The back-propagation algorithm was implemented in the *backprop.lisp* file in Appendix D. The function *train* takes a network object, *input-vector*, and *acceptable-error* as parameters. *Train* calls *train-set* with the network object and the *input-vector* until the error is within the *acceptable-error* level.

```
(defun train (my-network input-vector acceptable-error)
  (do* ((error (train-set my-network input-vector)
                    (train-set my-network input-vector)) )
        ((< error acceptable-error) 'done))
```

The *input-vector* consist of each of the training sets input data followed by the expected output for that particular run through the network. The input vector in this network consist of a list containing 5 list. Each of the five list are made up of two list containing the variable refering to the input data for that round and the expected output for that round. For instance, one such input vector would look like this:

```
((81mm(1 0 0 0 0))(60mm(0 1 0 0 0))(arty(0 0 1 0 0))(heat(0 0 0 1 0))(3-5in(0 0 0 0 1)))
```

Train-set trains the network on the *input-vector* by calling *train-network* iteratively with each of the five list in the *input-vector*. *Compute-error* computes the error for each round. The errors for each round, *current-error*, are added to *total-error* and compared to the *acceptable-error* in *train* to determine when to stop training the network.

```
(defun train-set (my-network input-vector)
  (do* ((i 0 (+ i 1))
        (current-error (compute-error
                        (train-network my-network (first (first input-vector))
                                         (second (first input-vector))))
        (if (nth i input-vector)
            (compute-error (train-network my-network
                                         (first (nth i input-vector))
                                         (second (nth i input-vector)))))
        (total-error current-error
                      (if (nth i input-vector)
                          (+ total-error current-error))))
        ((> i (- (length input-vector) 2)) total-error)))
```


Train-network activates the network with one of the rounds and computes the *output-error-vector* by calling *calc-output-error* with the *expected-output* and the *output-vector* of the network. The *hidden-layer error-vector* is calculated by calling *hidden-layer-errors* with the *output-error-vector* and the *node-list* in reverse order. The *node-list* of the network is the layers. By reversing the *node-list*, the layers are iterated through from output layer to input layer in the function *hidden-layer-errors*. The function *hidden-layer-errors* will be discussed later. Now that the *output-error-vector* and the *hidden-layer-error-vector* are known, the weights at each neuron can be changed. This is accomplished by calling *calculate-weight-change* with the *node-list* in reverse and a list containing the *output-error-vector* and the *hidden-layer-error-vector*. *Calculate-weight-change* is also an involved function and will be discussed later. At this point the *weight-vector* at the layer level has been changed and now the *weight-vector* for each neuron needs to be changed. *Set-neuron-weights* accomplishes this task.

```
(defmethod train-network ((my-network network) input-list
                          expected-output)
  (activate my-network input-list)
  (let* ((output-error-vector (calc-output-error expected-output
                                                  (output-vector my-network)))
        (hidden-layer-error-vector
         (hidden-layer-errors output-error-vector
                              (reverse (node-list my-network)))))
    )
  (calculate-weight-change (reverse (node-list my-network))
                           (cons output-error-vector
                                hidden-layer-error-vector))
  (set-neuron-weights (node-list my-network)
                      (output-vector (first (last (node-list my-network)))
                                     output-error-vector))
  ))
```

The function *hidden-layer-errors* calculates the incoming error of the hidden layer by calling *calc-inc-errors* with the *error-vector* and a transposed *weight-vector*. In effect what *calc-inc-errors* does is multiply the *output-error* caused by the neuron with the *weight-vector* and adds up the values in the resulting list. This value, called the *incoming-error* in the function *calc-final-error*, is fed back through the sigmoid function by calculating the derivative of the sigmoid function.

```
(defun hidden-layer-errors (error-vector layer-list)
  (do* ((i 0 (+ i 1))
        (new-error-vector
         (cons (calc-final-error
                  (calc-inc-errors error-vector
                                   (conv-weight-list (weight-vector (first layer-list)))
                                   (first layer-list))
                ())))
        (cons (calc-final-error
                  (calc-inc-errors error-vector
                                   (conv-weight-list (weight-vector (nth i layer-list)))
                                   (nth i layer-list))
                new-error-vector)) )
        ((> i (- (length layer-list) 2)) (reverse new-error-vector))))
```

```
(defmethod calc-final-error (incoming-error-vector layer)
  (let ((vector1 (input-vector layer)))
    (mapcar #'calculate-final-error vector1 incoming-error-vector))
)

(defun calculate-final-error (node-value incoming-error)
  (* incoming-error node-value (- 1 node-value)))

(defun calc-inc-errors (output-error-vector output-weights)
  (do* ((i 0 (+ i 1))
        (inc-error (cons (my-summation output-error-vector
                                         (first output-weights))
                          ())))
        (cons (my-summation output-error-vector
                          (nth i output-weights))
              inc-error))
  )
  ((> i (- (length output-weights) 2)) (reverse inc-error)) )
```

Calculate-weight-change iterates through each layer in the network calling *calc-wt-chg* with the *layer-list*, *error-vector*, and *weight-vector* for that layer. *Calc-wt-chg* calculates the amount of change by calling *calculate-delta-weight* with the learning rate (**step**) and the above parameters. *Calculate-delta-weight* multiplies the **step** with *error* and the *arc-weight* (original value at the neuron). *Calc-wt-chg* then calls *change-weight* which actually changes the weights for that layer. It is not easy to see what is actually going on here, but it follows the back-propagation algorithm laid out in Chapter III.

```
(defun calculate-weight-change (layer-list error-vector-list)
  (if (first layer-list)
      (setf(weight-vector (first layer-list))
            (calc-wt-chg (first layer-list) (first error-vector-list)
                          (weight-vector (first layer-list))) )
      (if (first layer-list)
          (calculate-weight-change (cdr layer-list) (cdr error-vector-list))))

  (defun calc-wt-chg (my-layer error-vector weight-vector-list)
    (do* ((i 0 (+ i 1))
          (weight1 (cons
                    (change-weight
                     (calculate-delta-weight *step*
                                               (first error-vector) (input-vector my-layer)
                                               (first weight-vector-list)) ())
                    (cons (change-weight
                           (calculate-delta-weight *step*
                                                       (nth i error-vector) (input-vector my-layer)
                                                       (nth i weight-vector-list)) weight1)))
          ((> i (- (length error-vector) 2)) (reverse weight1))))
```

```
(defun calculate-delta-weight (beta-shift error arc-weight)
  (do* ((i 0 (+ i 1))
        (delta-weight (cons (* beta-shift error (first arc-weight)) ())
                          (cons (* beta-shift error (nth i arc-weight)) delta-weight))
        )
        ((> i (- (length arc-weight) 2)) (reverse delta-weight))
  ))

(defun change-weight (vector1 vector2)
  (mapcar #' + vector1 vector2))
```

5. User Interface

The user interface is in the *ammo-rec.lisp* file found in Appendix E. This file contains three driver functions that the user can call in order to run the ammunition neural network. The first is the *ammo-recognition* function which creates a new network by making a network object called *ammo-network*. It sets up the network parameters that are passed to *initialize-network* along with the network object. Next, the function calls *train* which invokes the back-propagation algorithm. Train continues to run until the acceptable level of error, which is passed to *train* is reached. Once the network has reached the acceptable level of error, the weights are written to a file called *ammo-wgts.dat*. This is how the network object is saved. After all, the only portion of the network that cannot be easily reproduced are the weights.

```
(defun ammo-recognition ()
  (defparameter *step* .2)
  (setf output-type-list '("81mm" "60mm" "105mm" "105heat" "3.5in"))
  (format t "In build-my-input ~%")
  (build-my-input)
  (setf ammo-parameters (list(list (length 81mm) 81mm ())
                               (list (length output-type-list) 81mm ())))
  (format t "In make-instance ~%")
  (setf ammo-network (make-instance 'network))
  (format t "In initialize ~%")
  (initialize-network ammo-network ammo-parameters)
  (format t "In train ~%")
  (train ammo-network (build-ammo-input) '.2)
  (output-file "data/ammo-wgts.dat")
  (save-weights ammo-network))
```

The second function that invokes the network is the *load-ammo-net* function. This function is used when the user wants to load a network that has already been trained.

The function *retrain* allows the user to continue training the network with current

```
(defun load-ammo-net (wgts)
  (setf output-type-list '("81mm" "60mm" "105mm" "3.5in" "105heat"))
  (build-my-input)
  (setf parameters (list (list (length t3-5in) t3-5in (first wgts))
                          (list (length output-type-list) t3-5in (second wgts))))
  (setf ammo-network (make-instance 'network))
  (initialize-network ammo-network parameters))
```

weights and a different *step* value, as well as a different level of *acceptable-error*. The *step* value is the learning rate of the network.

```
(defun retrain-ammo-net (wgts)
  (defparameter *step* .1)
  (setf output-type-list '("81mm" "60mm" "105mm" "3.5in" "105heat"))
  (build-my-input)
  (setf parameters (list (list (length t3-5in) t3-5in (first wgts))
```

The user can modify the *ammo-net* function that activates the network. The *ammo-net* function sends the test input to activate and outputs the type of round the net decides the input resembles.

```
(defun ammo-net()
  (format t "input: 60mm output: ~A~% "
    (determine-output(activate ammo-network tst60mm)))
  (format t "input: 81mm output: ~A~% "
    (determine-output(activate ammo-network tst81mm)))
  (format t "input: 105mm arty output: ~A~% "
    (determine-output(activate ammo-network tst105mm)))
  (format t "input: 3.5in rocket output: ~A~% "
    (determine-output(activate ammo-network tst3-5in)))
  (format t "input: 105mm heat output: ~A~% "
    (determine-output(activate ammo-network tstheat))))
```

This program was also written in C++ to improve performance by reducing training time. The next chapter summarizes the findings and lessons learned in this thesis.

V. RESULTS

A. AMMUNITION GRAPHS

The actual data values collected on each round can be found in appendix E. The 31 x 35 matrix of data values is a bit overwhelming, so all of the data collected on the rounds was graphed using MATLAB. The graphs show both the similarities and differences in the rounds. Figures 5.1 - 5.5 contain the graphs of the ammunition. As you can see, the 105mm heat round and the 81mm mortar graphs are virtually identical. This fact will explain the results of the testing phase.

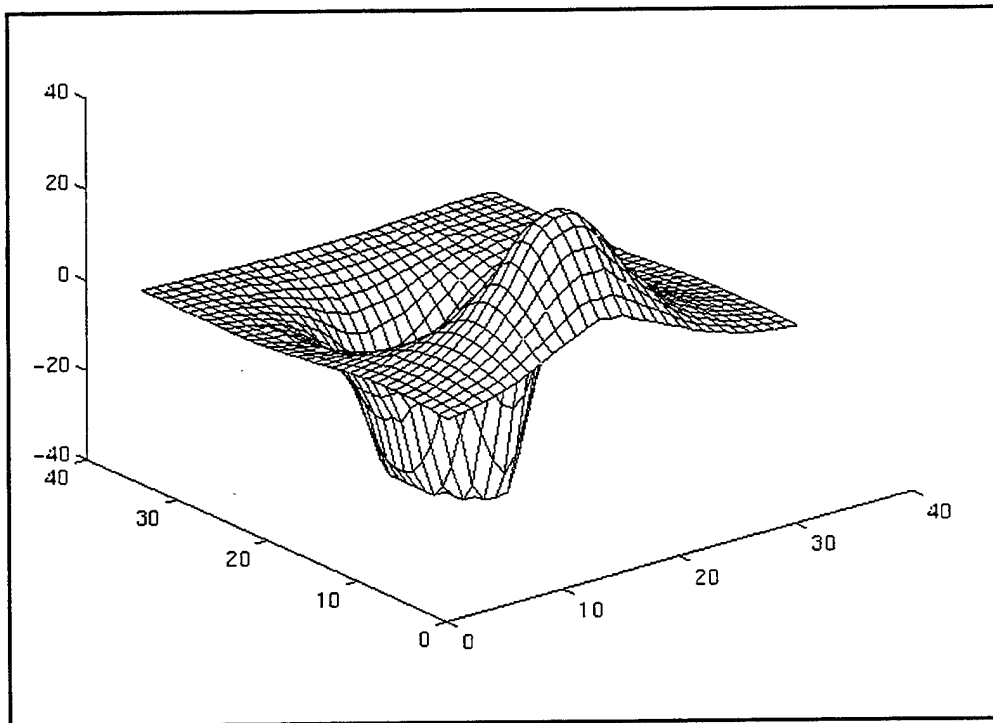


Figure 5.1: 60mm mortar input data graph.

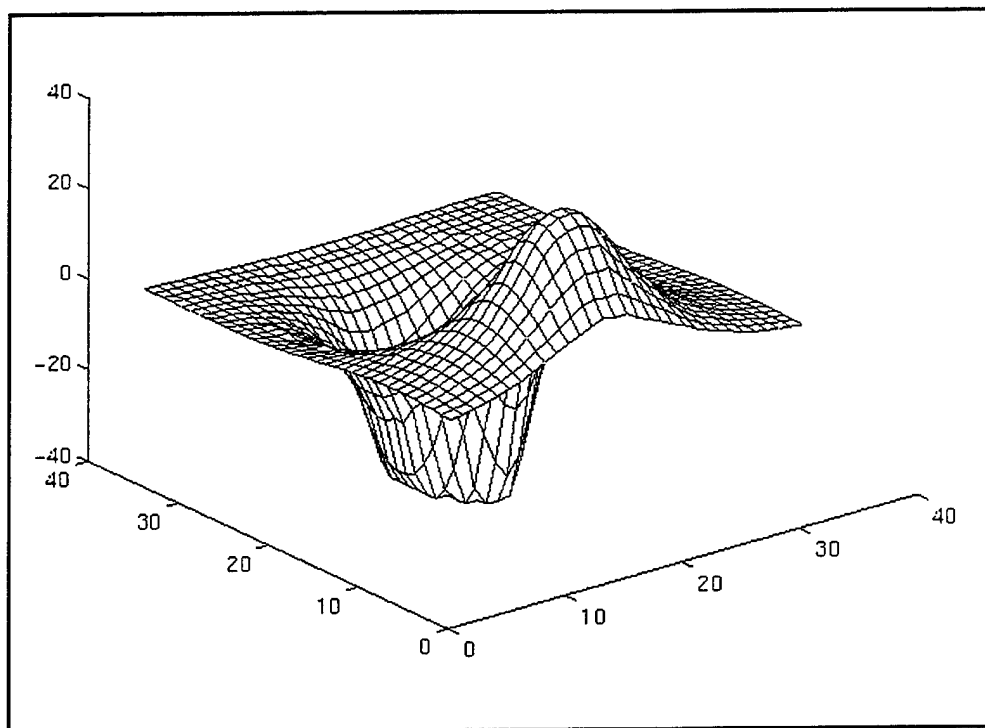


Figure 5.2: 81mm mortar input data graph.

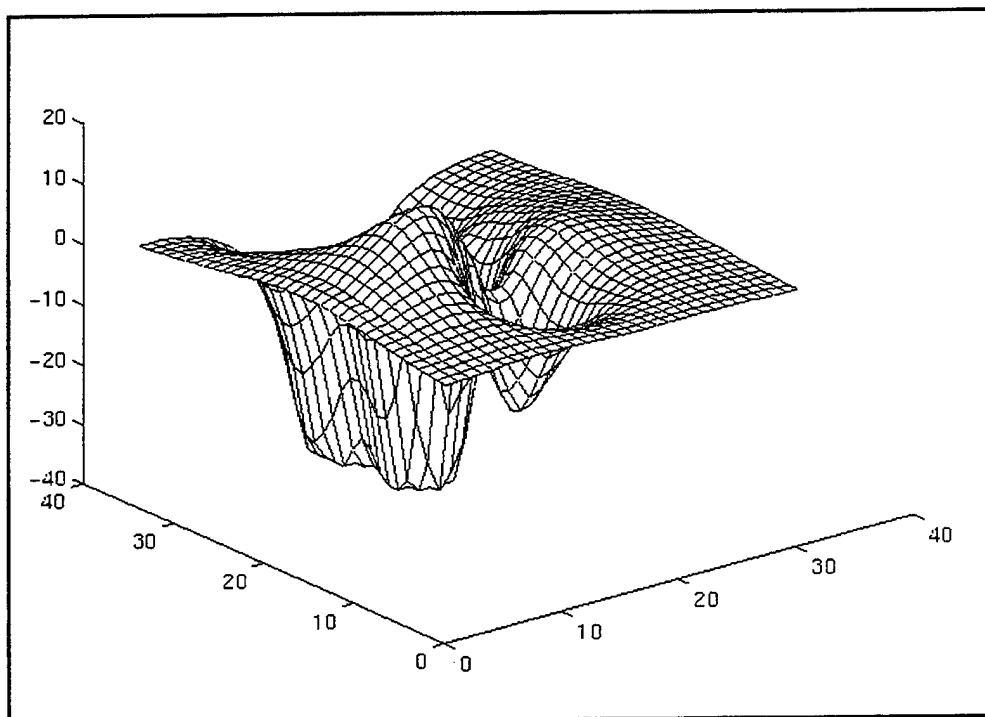


Figure 5.3: 105mm artillery input data graph.

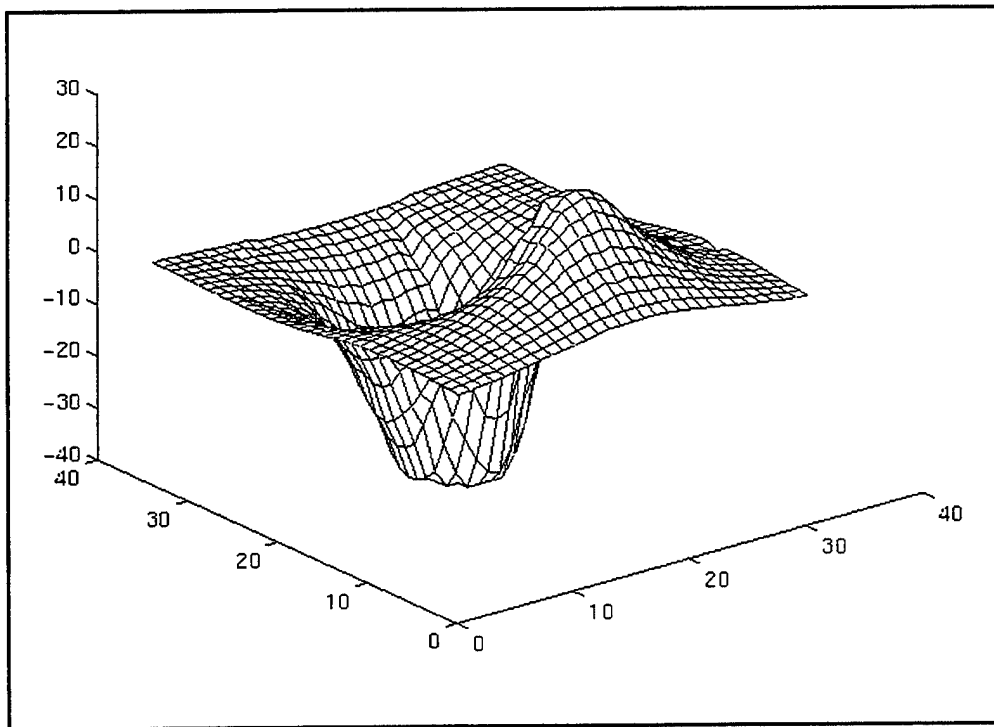


Figure 5.4: 105mm HEAT input data graph.

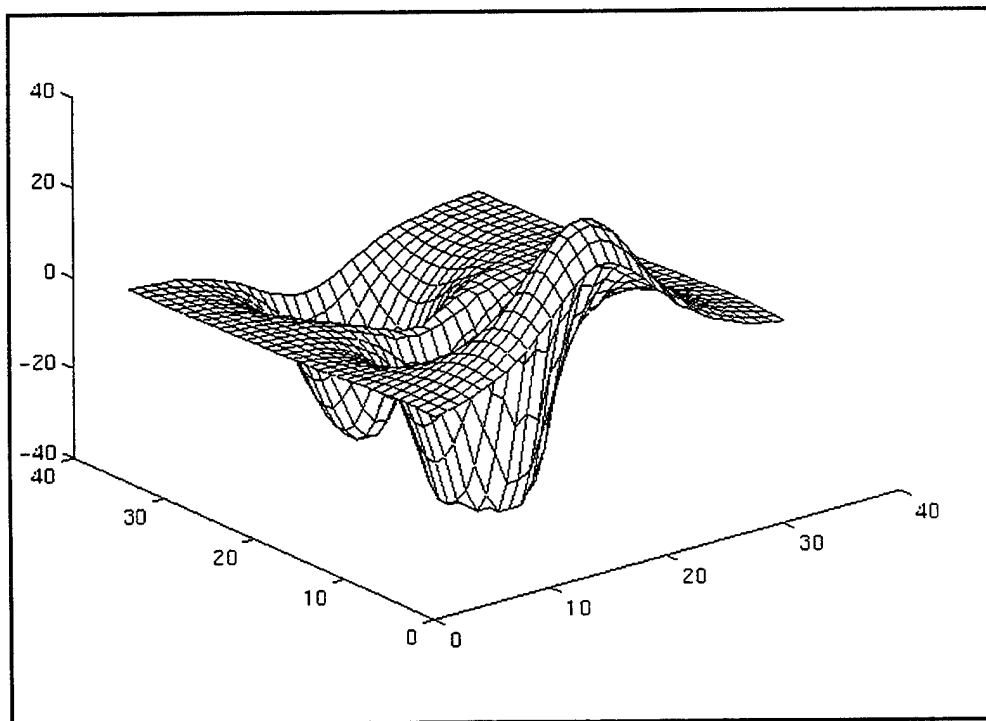


Figure 5.5: 3-5in rocket input data graph.

B. TRAINING

The amount of time a network takes to train is directly related to the values of the starting weights, learning rate and acceptable error. If the starting weights are close to the ideal weights then the time to train the network is less. If the learning rate is too small the network will take a long time to reach the optimal weights. If the learning rate is too large then the error will jump back and forth across the bowl (Figure 3.3). The acceptable level of error is the stopping point, so it will directly effect the time a network takes to train. All of these factors, along with using Lisp, make it hard to judge the performance of the back propagation algorithm.

The current network took 64 hours to train on an IRIX System V.4. Random weights, a learning rate of .2 and an error rate of .2 were used. Only 13 iterations of feed forward and back propagation were executed. The training phase produce 13.1 MB of data containing the weights for the neural network.

From these figures it is obvious that Lisp took a lot of the time to declare variables, dynamically allocate memory, compute the calculations and make the changes in the weights. This neural network would not work on a Sun OS due to the unavailability of enough heap space. The above IRIX OS release 5 with four 40 MHZ processors and 98 MB of memory was used to train the network. The network was originally written in a recursive format due to the nature of Lisp, but it had to be rewritten iteratively in order to have enough heap space to run. The nature of the neuron objects also carried a lot of overhead with them. Each neuron object has a weight vector of 1085 values and there are 1095 neuron objects. These figures translate into 1.18×10^6 values that have to be represented as variables. A network with a relatively large input set would run faster if it were written as functions, multiplying matrices, the least amount of overhead, the better.

Figures 5.6 and 5.7 show a graph of the error values after each iteration when the ammunition network was trained. The graph displays the error values on the left and the

iterations along the bottom. Notice how the iterations jump back and forth across the bowl. Remember a .2 error and .2 learning rate was used to train this network.

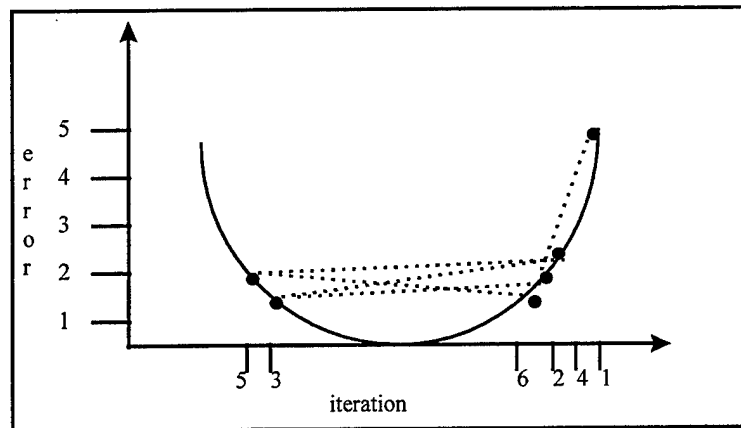


Figure 5.6: Graph of error convergence for iterations 1-6.

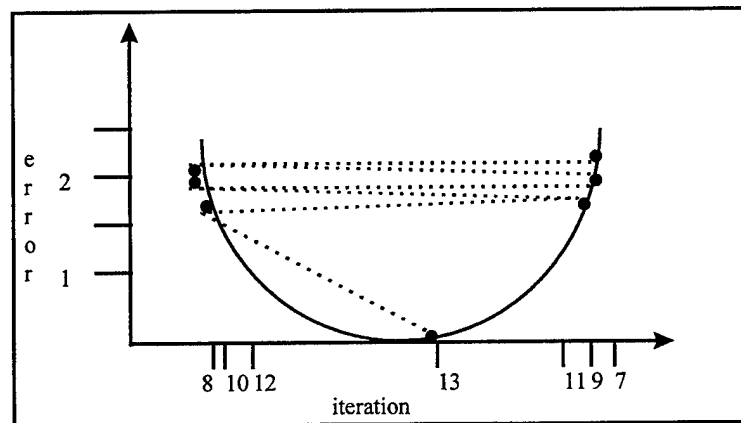


Figure 5.7: Graph of error convergence for iterations 7-13.

C. TESTING

Once the neural net was trained, the next step was to test the network on data it had not seen before. A third set of data was collected on the second training set of ammunition. These readings were different from the original training set which ensured the network had not seen the data before. To say a network has seen data before means it has been trained on the data. The following results point out the type of ammunition fed to the network, the amount of time it took to come up with the answer, the amount of memory used and how the network classified the ammunition. The order in which the output list

classifies the ammunition is 81mm mortar, 60mm mortar, 105mm artillery, 3-5in rocket, and 105mm HEAT. For instance, the 0.0 in (0.0 6.91607e-26 1.59656e-8 1.0 6.895369e-13) is the 81mm mortar, 6.91607e-26 is the 60 mm mortar, 1.59656e-8 is the 105mm artillery round, 0.0 is the 3-5inch rocket, and 6.895369e-13 is the HEAT round. The 1.0 shows that the network decided the input data was a 3-5in rocket. All of the other numbers are essentially zero due to the negative exponents.

- 3-5in rocket

```
[3] user(11): (time (activate ammo-network tst3-5in))
; cpu time (non-gc) 40,790 msec user, 390 msec system
; cpu time (gc)      10,380 msec user, 40 msec system
; cpu time (total)  51,170 msec user, 430 msec system
; real time  51,695 msec
; space allocation:
4,742,615 cons cells,; 0 symbols, 37,968,408 other bytes
(0.0 6.91607e-26 1.59656e-8 1.0 6.895369e-13)
```

- 105mm artillery

```
[3] user(12): (time (activate ammo-network tst105mm))
; cpu time (non-gc) 40,980 msec user, 270 msec system
; cpu time (gc)      9,120 msec user, 40 msec system
; cpu time (total)  50,100 msec user, 310 msec system
; real time  50,877 msec
; space allocation:
4,742,592 cons cells,; 0 symbols, 37,966,920 other bytes
(0.0 0.0 1.0 0.026313068 0.8945341)
```

- 60mm mortar

```
[3] user(13): (time (activate ammo-network tst60mm))
; cpu time (non-gc) 41,330 msec user, 80 msec system
; cpu time (gc)      11,620 msec user, 70 msec system
; cpu time (total)  52,950 msec user, 150 msec system
; real time   53,547 msec
; space allocation:
  4,742,592 cons cells,;  0 symbols, 37,966,920 other bytes
(0.0 1.0 0.011034517 3.4060365e-8 4.8678684e-9)
```

- 81mm mortar

```
[3] user(14): (time (activate ammo-network tst81mm))
; cpu time (non-gc) 41,590 msec user, 80 msec system
; cpu time (gc)      9,200 msec user, 50 msec system
; cpu time (total)  50,790 msec user, 130 msec system
; real time   51,062 msec
; space allocation:
  4,742,592 cons cells,;  0 symbols, 37,966,920 other bytes
(5.8375394e-14 1.521067e-13 4.5806116e-13 4.017225e-10 0.99741983)
```

- 105 HEAT

```
[3]user(15):(time (activate ammo-network tstheat))
;cpu time(non-gc)41,360 msec user, 60 msec system
;cpu time (gc)    12,480 msec user, 60 msec system
;cpu time(total) 53,840 msec user, 120 msec system
; real time   54,085 msec
; space allocation:
  4,742,592cons cells,;0symbols,37,966,920other bytes
(3.7480947e-29 7.8907937e-15 3.702611e-11 3.8968346e-15 1.0)
```

The average of the real time taken to compute an answer after training is 52.253 seconds. The amount of memory used was 42.7 MB. The test phase shows that 4 out of 5 of the rounds were correctly identified. In the test data, the 81mm mortar was the only round that was not correctly identified. It was identified as a 105mm HEAT round with .99 out of 1.0 accuracy. However all of the other pieces of ammunition were correctly identified with a precision of 1.0. A look at the graph of the 81mm mortar and the 105mm HEAT rounds will show a remarkable similarity. The results might have been better if there were not max readings of -36.0 and 36.0. The HEAT round would have had a higher peak than the 81mm mortar. When the training data was feed through the network all of the ammunition was correctly classified. The 81mm mortar was classified as an 81mm mortar.

Further testing revealed that when data the network had not seen before and was not trained to identify was input, the network gave false positives. Readings were taken on a tin can and the network identified the can as an artillery round with a 1.0 accuracy. This result was an unfortunate turn of events. Hopefully, the network would have came up with low numbers, .70 or less, on all of the outputs for the tin can. An analysis of the graph of the tin can and the artillery shell revealed that the two sets of input data were similar but not identical. Figure 5.8 shows the graph of the tin can.

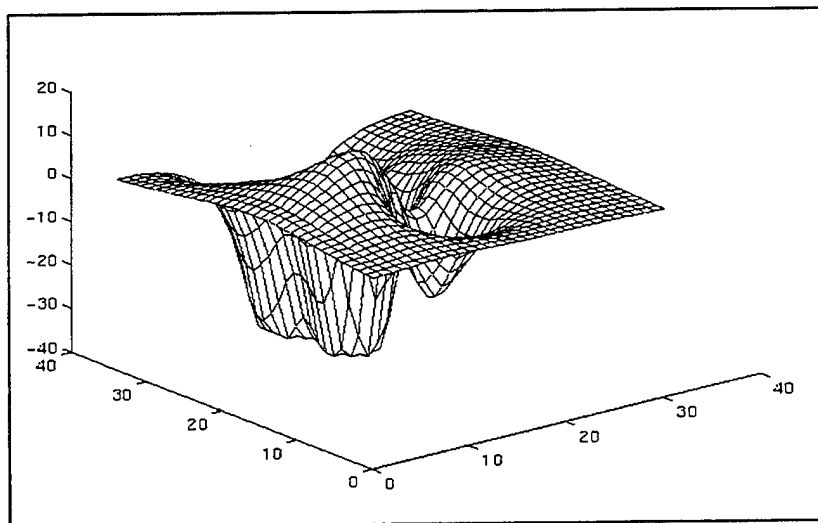


Figure 5.8: Tin can input data graph.

VI. CONCLUSIONS

A. THESIS QUESTIONS

The goal of this thesis was to determine if an artificial neural network was capable of correctly identifying unexploded ordnance. In making this determination the following thesis questions were addressed:

- Are artificial neural networks able to correctly identify, within a certain degree of precision, various type of unexploded ordnance both surface laid and buried?
- What type of neural network architecture is best for the job?
- What is the training set to be used in the training of the neural network?
- With what precision are the objects correctly identified?

Are artificial neural networks able to correctly identify, within a certain degree of precision, various type of unexploded ordnance both surface laid and buried? It has been shown that a neural network is capable of identifying 4 out of the 5 pieces of ammunition with a .99 or higher certainty. These results are based on data inputs from a test set made up of a 60mm mortar, 81mm mortar, 105mm artillery round, 105mm HEAT round and a 3-5in rocket. The amazing part of this result is that only two sets of data on the above ammunition were used to train the network. A neural network is trained to recognize a pattern. The more variations of that pattern used to train the network, the more efficient the network is in correctly classifying a pattern that is slightly different from the pattern used to train the network. Yet, the multi-layer feed-forward artificial network was able to correctly identify 4 pieces of ammunition with only two sets of training data.

The quality of the sensor and the limited range of output values lead to the failure of the network to correctly identify the 81mm mortar. The range of the sensor is -36.0 to 36.0. Both the 81mm mortar and the 105mm HEAT round reached the maximum value of the sensor in the same area of the grid. If the sensor's maximum value was higher, the

two rounds would have different values in the center of the grid. Therefore, the network would be able to distinguish between the two rounds.

When the two sets of training data were input in to the network, 100% of the ammunition was correctly identified. This fact also leads me to believe that with more training data the network would be able to distinguish between the 81mm mortar and the 105mm HEAT round of the test set.

Where as the test set results are encouraging, the failure of the network to not identify the tin can is an area of concern. However, when the network was retrained with the tin can included in the training set to produce all zeros, the network trained. This shows that if false positives come up on certain types of objects, the network can be trained to not recognize the objects. This may allow a neural network to be constructed that could be a useful tool in clearing ranges.

Only surface laid ammunition was tested in this thesis. The problem with a network correctly identifying buried ammunition is one of limiting the number of possibilities. There are an infinite number of possible ways the piece of ammunition could be buried in the ground. The number of possibilities span from the depth of the round to the angle of the round. Each of these possibilities will yield a different set of input data. Therefore, the problem becomes one of reducing the above infinite set to a finite set by limiting the number of angles and depths at which readings are taken. Another more viable options is to develop a mathematical equation that would normalize the data by bringing the input readings of a buried round to a base case of the above surface laid readings. This equation would be based on finding the centroid of the magnetometer readings of the round and then orienting the round based upon the strength of the readings.

What type of neural network architecture is best for the job? A multi-layered feed-forward neural network with the back propagation training algorithm was used for the ammunition recognition network. This network was chosen due to its capabilities as a pattern classification network. Execution speed and the number of layers were also a

major determining factors in choosing an architecture. Once a feed-forward network is trained, the actual execution time is very fast. With 1085 neurons in the input layer, keeping the number of layers to a minimum also contributes to the execution time.

What is the training set to be used in the training of the neural network? As mentioned above, a training set consisting of over 1000 data points was collected for 2 sets of a 60mm mortar, 81mm mortar, 105mm artillery round, 105mm HEAT round, and a 3-5in rocket. These rounds were chosen because they are common types of UXO's and because of their availability at a local ordnance unit.

The last thesis question of what precision the objects are correctly identified has already been addressed. Four of the five rounds were correctly identified with a precision of .99 or better with only two training sets of data.

B. LESSONS LEARNED

Gathering the data was the most time consuming process of this research. This step must be automated. Any network improves in performance with the amount of training data. This network was trained on only two sets of the ammunition set. Ten to 100 or even 1000 sets of data would have been much more effective. Using a large training set is a normal practice. The greater the number of training sets of different data on the same object, the better any neural network will perform. Automating this data collection process using the Naval Postgraduate School autonomous vehicle, Shepard, would have made gathering more data possible.

The position of the round when gathering data is critical to the success of the network. Moving the round up, down, left or right, will cause the network to not recognize the pattern. This problem leads back to the gathering of more data. The more data gathered on the training set in different positions, the better chance the network has to identify the test set of ammunition. This is critical when it comes to sending Shepard out in the field to identify UXO's. More than likely, the round will not be in the same position as it was when the network was trained.

Using LISP as the language for the prototype was very helpful. But in order for the network to achieve the level of speed required of a real time system, the program must be rewritten in C++ or implemented in hardware. Hardware is the better solution. I rewrote the network in C++ in order to achieve a faster training time (see Appendix G). Training time in LISP was taking hours to iterate through one piece of ammunition in the training set. The C++ version was written with as little overhead as possible and was able to iterate through the entire training set in a matter of minutes, which was an order in magnitude improvement.

C. RECOMMENDATIONS FOR FUTURE RESEARCH

The most important recommendation for future research is to mount the array of sensors on Shepard. This will allow for the automated gathering of a large amount of training data. I recommend the use of a one dimensional array of 31 sensors on a platform out in front of Shepard. The further in front of Shepard this array can be placed the better, because Shepard is made of metal and this may affect the sensors.

Future research must include the ability to determine the identity of a buried object. Automating the data collection will aid in the development of a method of determining what type of rounds are buried in the ground. Whether the decision is made to gather an exhaustive amount of data to train the network on or develop an equation to determine the identity of a buried object, automating the data collection is essential.

The concept of using an artificial neural network to identify UXO's is a valid concept and should be pursued further. Although I was unable to prove that a neural network will identify a UXO from a NON-UXO, the fact that the network can identify one UXO from another is a very significant finding. I believe there exists a set of weights that will make the neural network distinguish a UXO from a NON-UXO. The only way this will be proven is the collection of more data through automating the process.

APPENDIX A: SOURCE CODE (NEURON CLASS)

```
;;-----  
;;File: neuron.lsp      Franz Common Lisp  
;;Jeff May  
;;21 Mar 97  
;;Contains the neuron class  
;; Copyright © 1997 Jeff May  
;;-----  
;;-----  
;;NEURON CLASS  
;;creates a neuron class with a weight value list, and output value list.  
;;-----  
(defclass neuron ()  
  ((weight-value :accessor weight-vector  
    :initarg :weight-value  
    :initform '())  
   (output-value :accessor output-vector  
    :initarg :output-value  
    :initform '(1))))
```

```

;;-----
;;NEURON ACTIVATION
;;Feeds the weighted sum of the input vector to the activation
;;function inorder to produce an output for the neuron.
;;-----
(defmethod neuron-activation ((my-neuron neuron) layer-input)
  (setf (output-vector my-neuron)
        (list (sigmoid (my-summation (weight-vector my-neuron)
                                     layer-input))))
  ))
;;-----
;;INITIALIZE NEURON
;;Takes an input vector and weight vector and sets the data members
;;of the neuron. If a nil weight vector is passed in, random weights
;; are assigned.
;;-----
(defmethod initialize-neuron ((my-neuron neuron) input-length weight-vector1)
  (if (null weight-vector1)
      (setf(weight-vector my-neuron) (make_random_weights input-length 2.0))
      (setf(weight-vector my-neuron) weight-vector1))
  )

```

```

;;-----*
;; SUMMATION OF A VECTOR
;; this does summation on a vector
;; > (summation vector)
;; > (summation '(2.0 3.0 2.6))
;;-----*
(defun my-summation (vector1 vector2)
  (apply '+ (mapcar #'* vector1 vector2)))
;;-----*
;; CREATE RANDOM WEIGHTS
;; makes a weight list of n elements long
;; of random weights +/- of rsw
;; > (make_random_weights 3 2.0)
;; (0.673233 -1.875556 1.333498)
;;(setf rsw 2.0);;or set it globally to a value
;;-----*
(defun list_of (n elt)
  (do* ((i 1 (+ i 1))
        (weight1 (cons elt ()) (cons elt weight1))
        )
    ((> i n) weight1)))

```

```

;;-----*
;;Make random weights
;;-----*
(defun make_random_weights (n rsw)
  (mapcar #'-
    (list_of n rsw)
    (mapcar #'random (list_of n (* rsw 2)))))
;;-----*
;; LOGISTIC ACTIVATION FUNCTION
;; this provides a standard activation
;; function for a neural net--the logistic
;; sigmoid function
;;  $f(x) = 1 / (1 + e^{-x})$ 
;;-----*
(defun sigmoid (x)
  (/ 1 (+ 1 (exp (- 0 x)))))

```

APPENDIX B: SOURCE CODE (LAYER CLASS)

```
;;-----  
;;layer class      Franz Common Lisp  
;;Jeff May  
;;21 March 97  
;;contains layer class  
;; Copyright © 1997 Jeff May  
;;-----  
;;-----  
;;Creates a layer class with a node-value, input-value, number and slot values inherited from  
;;neuron class. Number is the number of neurons in the layer. Node-value is a list containing  
;;the neurons.  
;;-----  
(defclass layer (neuron)  
  ((node-value :accessor node-list  
    :initarg :node-value  
    :initform '((make-instance 'neuron)))  
   (input-value :accessor input-vector  
    :initarg :input-value  
    :initform '())  
   (number :accessor number-value  
    :initarg :number  
    :initform 1)))
```

```

;;-----
;;INITIALIZE LAYER
;;Initializes the slot values of layer class.
;;-----

(defmethod initialize-layer ((my-layer layer) layer-params)
  (setf(number-value my-layer) (first layer-params))
  (setf(input-vector my-layer) (second layer-params))
  (setf(weight-vector my-layer) (third layer-params))
  (setf(node-list my-layer) (build-layer (first layer-params)))
  (initialize-node-list my-layer layer-params)
)

;;-----
;;INITIALIZE NODE LIST
;;Initializes the neurons in the node list.
;;-----

(defmethod initialize-node-list ((my-layer layer) layer-params)
  (do* ((i 0 (+ i 1))
        (neuron1 (first (node-list my-layer)) (nth i (node-list my-layer)))
        (weight1 (first (third layer-params))
                  (if (null (third layer-params))
                      (first (third layer-params))
                      (nth i (third layer-params)))))
    )
    (> i (- (length (node-list my-layer)) 1)))
  (initialize-neuron neuron1 (length (input-vector my-layer)) weight1)))

```



```

;;-----
;;BUILD LAYER
;;Creates the number of neurons need for the layer
;;-----
(defun build-layer (number-neurons)
  (do* ((i 1 (+ i 1))
        (neuron-list (cons (make-instance 'neuron) ()))
        (cons (make-instance 'neuron) neuron-list)))
    ((> i (- number-neurons 1)) (reverse neuron-list))))
;;-----
;;ACTIVATE LAYER
;;Sets the weight-value slot by activating the layer
;;-----
(defmethod activate-layer ((my-layer layer))
  (setf(weight-vector my-layer) (set-weights (node-list my-layer)))
  (setf(output-vector my-layer) (activate-layer-list my-layer))
  )
;;-----
;;ACTIVATE LAYER LIST
;;Activates the neurons in the layer by calling neuron-activation
;;-----
(defmethod activate-layer-list ((my-layer layer))
  (do* ((i 0 (+ i 1))
        (layer1 (cons (first (neuron-activation
                               (first (node-list my-layer)) (input-vector my-layer)))) ()))
    )

```

```

      (cons (first (neuron-activation (nth i (node-list my-layer))
                                     (input-vector my-layer))) layer1))
    )
    ((> i (- (length (node-list my-layer)) 2)) (reverse layer1))
  ))
;;-----
;;SET WEIGHTS
;;If weights are sent in as parameters then is function is called instead of random weights.
;;-----
(defun set-weights (layer-list)
  (if (null layer-list)
      ()
      (cons (weight-vector (car layer-list))
            (set-weights (cdr layer-list))))))

```

APPENDIX C: SOURCE CODE (NETWORK CLASS)

```
;;-----  
;;network.lsp Franz Common Lisp  
;;Jeff May  
;;21 March 97  
;;Contains network class  
;; Copyright © 1997 Jeff May  
;;-----  
;;-----  
;;Creates a network class which is an instance of a layer class.  
;;-----  
(defclass network (layer)  
  ((nodes-layer-value :accessor nodes-per-layer  
    :initarg :nodes-layer-value  
    :initform '(1))))  
;;-----  
;;INITIALIZE NETWORK  
;;initializes the slot values of network class based on the parameters passed in.  
;;-----  
(defmethod initialize-network ((my-network network) network-params)  
  (setf(input-vector my-network) (second (first network-params)))  
  (setf(number-value my-network) (length network-params))  
  (setf(nodes-per-layer my-network) (list (first (first network-params))  
    (first (second network-params))))  
  (setf(node-list my-network) (build-network
```

```

                (length network-params)))

(initialize-network-layers my-network network-params)

)

;;-----
;;INITIALIZE NETWORK LAYERS
;;Initializes each layer in the network by calling initialize-layer
;;-----

(defmethod initialize-network-layers ((my-network network) network-params)

  (do* ((i 0 (+ i 1))

        (layer1 (first (node-list my-network)) (nth i (node-list
                                                         my-network))))

    )

    ((> i (- (length (node-list my-network)) 1)))

    (initialize-layer layer1 (nth i network-params))))

)

;;-----
;;BUILD NETWORK
;;Creates the layers in the network
;;-----

(defun build-network (number-layers)

  (do* ((i 1 (+ i 1))

        (layer-list (cons (make-instance 'layer) ()))

        (cons (make-instance 'layer) layer-list)))

    ((> i (- number-layers 1)) (reverse layer-list))))

)

```

```

;;-----
;;ACTIVATE
;;sets the output-value of the network to the value of the last layer output-value by calling
;;activate-network
;;-----
(defmethod activate ((my-network network) my-network-input)
  (set-network-input my-network my-network-input)
  (activate-network my-network)
  (setf(output-vector my-network)
    (output-vector (first (last (node-list my-network))))))
  )
;;-----
;;SET NETWORK INPUT
;;Sets network input vector
;;-----
(defmethod set-network-input ((my-network network) network-input)
  (setf(input-vector my-network) network-input)
  (setf(input-vector (first (node-list my-network))) network-input)
  )
;;-----
;;ACTIVATE NETWORK
;;Activates network by sending one layer at a time to initialize-layer
;;-----
(defmethod activate-network ((my-network network))
  (do* ((i 0 (+ i 1))

```

```

(output-layer (activate-layer (first (node-list my-network)))
  (if (nth i (node-list my-network))
    (activate-layer (nth i (node-list my-network)))))
)
((> i (- (length (node-list my-network)) 1)))
(if (nth (+ i 1) (node-list my-network))
  (setf(input-vector (nth (+ i 1) (node-list my-network)))
    output-layer))))

```

APPENDIX D: SOURCE CODE (BACK PROPAGATION)

```
;;-----  
;;backprop.lsp      Franz Common Lisp  
  
;;Jeff May  
  
;;21 Mar 97  
  
;;Contains the functions that run the back propagation algorithm  
  
;; Copyright © 1997 Jeff May  
  
;;-----  
;;-----  
;;TRAIN  
;;Repeat process until total error is within acceptable level  
;;-----  
(defun train (my-network input-vector acceptable-error)  
  (do* ((error (train-set my-network input-vector)  
               (train-set my-network input-vector))  
        )  
    ((< error acceptable-error) 'done)  
    (format t "Total Error ~A~% " error)))  
;;-----  
;;TRAIN-SET  
;;Total Error = Total Error + Total Output Error (current-error)  
;;-----  
(defun train-set (my-network input-vector)  
  (do* ((i 0 (+ i 1))
```

```

(current-error (compute-error
                (train-network my-network (first (first input-vector))
                (second (first input-vector))))
(if (nth i input-vector)
    (compute-error (train-network my-network
                                (first (nth i input-vector))
                                (second (nth i input-vector)))
    )))
(total-error current-error
(if (nth i input-vector)
    (+ total-error current-error)))
)
((> i (- (length input-vector) 2))total-error)
))

```

```
;;-----
```

```
;;COMPUTE ERROR
```

```
;;Total Output Error = sum(output errors)
```

```
;;-----
```

```
(defun compute-error (error-vector)
```

```

  (let* (
    (vector (mapcar #'abs error-vector))
    (sum (mapcar #'+ vector))
    ) (first sum))
  )

```



```

;;-----
;;TRAIN NETWORK
;;-----

(defmethod train-network ((my-network network) input-list expected-output)
  (activate my-network input-list)
  (let* ((output-error-vector (calc-output-error expected-output
                                         (output-vector my-network)))
         (hidden-layer-error-vector
          (hidden-layer-errors output-error-vector (reverse (node-list
                                                             my-network))))
        )
    (calculate-weight-change (reverse (node-list my-network))
                             (cons output-error-vector
                                   hidden-layer-error-vector))
    (set-neuron-weights (node-list my-network))
    (output-vector (first (last (node-list my-network))))
    output-error-vector
  ))

;;-----
;;CALC-OUTPUT-ERROR
;;Output Error = Expected - Actual
;;-----

(defun calc-output-error (exp-output calc-output)
  (mapcar #'- exp-output calc-output))

```

```

;;-----
;;HIDDEN LAYER ERRORS
;;
;;-----
(defun hidden-layer-errors (error-vector layer-list)
  (do* ((i 0 (+ i 1))
        (new-error-vector
         (cons (calc-final-error
                 (calc-inc-errors error-vector
                                   (conv-weight-list (weight-vector (first layer-list)))
                                   )
                 (first layer-list))
                ()))
        (cons (calc-final-error
                 (calc-inc-errors error-vector
                                   (conv-weight-list (weight-vector (nth i layer-list)))
                                   )
                 (nth i layer-list))
              new-error-vector)
        )
        )
  ((> i (- (length layer-list) 2)) (reverse new-error-vector))))

```

```

;;-----*
;; CALC-FINAL-ERROR
;;-----*

(defmethod calc-final-error (incoming-error-vector layer)
  (let ((vector1 (input-vector layer)))
    (mapcar #'calculate-final-error vector1 incoming-error-vector)
  )
)

;;-----*
;; CALCULATE FINAL ERROR
;;FinErr = IncErr * Derivative_of_activation_function
;;-----*

(defun calculate-final-error (node-value incoming-error)
  (* incoming-error node-value (- 1 node-value)))

;;-----*
;; Calculate Incoming Errors
;;IncErr = sum(OutErr * weights)
;;-----*

(defun calc-inc-errors (output-error-vector output-weights)
  (do* ((i 0 (+ i 1))
        (inc-error (cons (my-summation output-error-vector
                                         (first output-weights))
                          ())
        (cons (my-summation output-error-vector
                              (nth i output-weights))
  )

```

```

        inc-error))

    )

    ((> i (- (length output-weights) 2)) (reverse inc-error)) )

;;-----
;;CALCULATE WEIGHT CHANGE
;;-----

(defun calculate-weight-change (layer-list error-vector-list)

  (if (first layer-list)

      (setf(weight-vector (first layer-list))

            (calc-wt-chg (first layer-list) (first error-vector-list)

                          (weight-vector (first layer-list))))

      )

  )

  (if (first layer-list)

      (calculate-weight-change (cdr layer-list) (cdr error-vector-list))

      )

  )

;;-----
;;CALC-WT-CHG
;;-----

(defun calc-wt-chg (my-layer error-vector weight-vector-list)

  (do* ((i 0(+ i 1))

        (weight1 (cons

                  (change-weight

                    (calculate-delta-weight *step*

```

```

        (first error-vector) (input-vector my-layer))
      (first weight-vector-list))
    ())
  (cons (change-weight
        (calculate-delta-weight *step*
          (nth i error-vector) (input-vector my-layer))
          (nth i weight-vector-list))
        weight1))
  )
  ((> i (- (length error-vector) 2)) (reverse weight1))
))

;;-----*
;; CALCULATE DELTA WEIGHT
;; delta_weight = b * OutErr * InputValues
;;-----*

(defun calculate-delta-weight (beta-shift error arc-weight)
  (do* ((i 0 (+ i 1))
        (delta-weight (cons (* beta-shift error (first arc-weight)) ()))
        (cons (* beta-shift error (nth i arc-weight)) delta-weight))
    )
    ((> i (- (length arc-weight) 2)) (reverse delta-weight))
  ))

```

```

;;-----*
;; CHANGE WEIGHT
;; weight = weight + delta_weight
;;-----*

(defun change-weight (vector1 vector2)
  (mapcar #' + vector1 vector2))

;;-----
;;SET NEURON WEIGHTS
;;-----

(defun set-neuron-weights (layer-list)
  (do* ((k 0 (+ k 1)))
    ((> k (- (length layer-list) 1)))
    (do* ((i 0 (+ i 1)))
      ((> i (- (length (node-list (nth k layer-list))) 1)))
      (setf(weight-vector (nth i (node-list (nth k layer-list))))
        (nth i (weight-vector (nth k layer-list))))
      )))

;;-----*
;; CONVERT WEIGHT_LIST TO BACKPROP_WEIGHT LIST
;; conv_weight_list
;; takes (1 2 3) (4 5 6) and makes (1 4) (2 5) (3 6)
;;-----

(defun conv-weight-list (x)
  (apply 'mapcar #'list x))

```

APPENDIX E: SOURCE CODE (AMMO-RECOGNITION)

```
;;-----  
;;ammo-rec.lsp  
;;Jeff May  
;;21 Mar 97  
;;Contains the user interface functions  
;; Copyright © 1997 Jeff May  
;;-----  
;;-----  
;;AMMO-RECOGNITION sets the learning rate, called step, to .2. Output-type-list establishes  
;;what the order of the output. Ammo-parameters is a list that contains the number of neurons  
;;in the first layer, length 81mm, weight list, (), initial input values, 81mm; the number of  
;;neurons in the nest layer, length output-type-list, initial input values for the second layer,  
;;81mm and weight vector for the second layer neurons, (). Setf ammo-network creates an  
;;network object called ammo-network.  
;;-----  
(defun ammo-recognition ()  
  (defparameter *step* .2)  
  (setf output-type-list ("81mm" "60mm" "105mm" "105heat" "3.5in"))  
  (format t "In build-my-input ~%")  
  (build-my-input)  
  (setf ammo-parameters (list(list (length 81mm) 81mm ())  
                               (list (length output-type-list) 81mm ())))  
  (format t "In make-instance ~%")  
  (setf ammo-network (make-instance 'network))  
  (format t "In initialize ~%")  
  (initialize-network ammo-network ammo-parameters))
```

```

(format t "In train ~%")

(train ammo-network (build-ammo-input) '.2)

(output-file "data/ammo-wgts.dat")

(save-weights ammo-network)

)

;;-----
;;LOAD-AMMO-NET
;;Loads a pre-existing network object and calls it ammo-network.
;;-----

(defun load-ammo-net (wgts)

  (setf output-type-list '("81mm" "60mm" "105mm" "3.5in" "105heat"))

  (build-my-input)

  (setf parameters (list (list (length t3-5in) t3-5in (first wgts))

                          (list (length output-type-list) t3-5in (second wgts))))

  (setf ammo-network (make-instance 'network))

  (initialize-network ammo-network parameters)

)

;;-----
;;RETRAIN-AMMO-NET
;;Allows for a method of retraining a network with the existing weights and a different learning
;;rate as well as a different error
;;-----

(defun retrain-ammo-net (wgts)

  (defparameter *step* .1)

  (setf output-type-list '("81mm" "60mm" "105mm" "3.5in" "105heat"))

```



```

(build-my-input)

(setf parameters (list (list (length t3-5in) t3-5in (first wgts))
                        (list (length output-type-list) t3-5in (second wgts))))

(setf ammo-network (make-instance 'network))

(initialize-network ammo-network parameters)

(train ammo-network (build-ammo-input) '.1)

(output-file "data/ammo-wgts.dat")

(save-weights ammo-network)

)

;;-----
;;AMMO-NET

;;Activates ammo-network, calls determine-output and prints what piece of ammo the net has
;;determined the input is and with what value. A value of 1.0 is the max. This function sends
;;the test data through the network.

;;-----

(defun ammo-net()

  (format t "input: 60mm output: ~A~% "

    (determine-output(activate ammo-network tst60mm)))

  (format t "input: 81mm output: ~A~% "

    (determine-output(activate ammo-network tst81mm)))

  (format t "input: 105mm arty output: ~A~% "

    (determine-output(activate ammo-network tst105mm)))

  (format t "input: 3.5in rocket output: ~A~% "

    (determine-output(activate ammo-network tst3-5in)))

  (format t "input: 105mm heat output: ~A~% "

    (determine-output(activate ammo-network tstheat))))

```

;;-----

;;BUILD-MY-INPUT

;;Creates the variables that contain the input data.

;;-----

```
(defun build-my-input ()  
  (setf 60mm (build-input "../data/60mma.dat"))  
  (setf 81mm (build-input "../data/81mm.dat"))  
  (setf 105mma (build-input "../data/105mmarty.dat"))  
  (setf 3-5in (build-input "../data/3-5in.dat"))  
  (setf 105heat (build-input "../data/105-heat.dat"))  
  (setf t3-5in (build-input "../data/t3-5in.dat"))  
  (setf t81mm (build-input "../data/t81mm.dat"))  
  (setf t105heat (build-input "../data/t105heat.dat"))  
  (setf t60mm (build-input "../data/t60mm.dat"))  
  (setf t105mm (build-input "../data/t105mm.dat"))  
  (setf tst60mm (build-input "../data/tst60mm.dat"))  
  (setf tst105mm (build-input "../data/tst105mm.dat"))  
  (setf tst3-5in (build-input "../data/tst3-5in.dat"))  
  (setf tstheat (build-input "../data/tstheat.dat"))  
  (setf tst81mm (build-input "../data/tst81mm.dat"))  
)
```

```
;;-----  
;;BUILD-AMMO-INPUT  
;;Builds the training sets with the expected output attached to each ammo variable
```

```
;;-----  
(defun build-ammo-input ()  
  (list  
    (list 81mm '(1 0 0 0 0))  
    (list 60mm '(0 1 0 0 0))  
    (list 105mma '(0 0 1 0 0))  
    (list 105heat '(0 0 0 1 0))  
    (list 3-5in '(0 0 0 0 1))  
    (list t81mm '(1 0 0 0 0))  
    (list t60mm '(0 1 0 0 0))  
    (list t105mm '(0 0 1 0 0))  
    (list t105heat '(0 0 0 1 0))  
    (list t3-5in '(0 0 0 0 1))) )
```

```
;;-----  
;;DETERMINE -OUTPUT  
;;Takes the max number of the output list and prints the corresponding string from the output-  
;;type-list.
```

```
;;-----  
(defun determine-output (output-list)  
  (let ((answer (eval (append '(max) output-list))))  
    (format t "~A~%" answer)  
    (nth(position answer output-list) output-type-list)))
```

```
;;-----  
;;ANN  
;;Compiles and loads all of the files needed to run the network  
;;-----  
(defun ann ()  
  (compile-file "neuron")  
  (load "neuron")  
  (compile-file "layer")  
  (load "layer")  
  (compile-file "network")  
  (load "network")  
  (compile-file "backprop")  
  (load "backprop")  
  (compile-file "ammo-rec")  
  (load "ammo-rec")  
  (compile-file "readnet")  
  (load "readnet")  
  (compile-file "out-data")  
  (load "out-data"))
```

```

;;-----
;;OUTPUT-FILE
;;Creates the output file for the weights of the network
;;-----

(defun output-file (file-name)
  (setf output-path (make-pathname :name file-name))
  (setf out-str (open output-path :direction :output
                      :if-exists :supersede))
  )
;;-----

;;SAVE-WEIGHTS
;;Writes the weights to the above file.
;;-----

(defmethod save-weights ((my-network network))
  (do* ((i 0 (+ i 1))
        (weight1 (weight-vector (first (node-list my-network))))
        (list (weight-vector (nth i (node-list my-network))) weight1))
    )
    ((> i (- (length (node-list my-network)) 2))
     (format out-str "~A~%" (reverse weight1)))
  )
  (close out-str))

```

```

;;-----
;;BUILD-INPUT
;;Opens the input file and reads the data in.
;;-----

(defun build-input (data-path)
  (setf path (make-pathname :name data-path))
  (setf str (open path :direction :input
                  :if-exists :supersede))
  (do* ((str-line (read-line str nil 'eof)
                  (read-line str nil 'eof))
        (ammo-list (list (read-from-string str-line))
                    (if (eql str-line 'eof)
                        ammo-list
                        (cons (read-from-string str-line) ammo-list))))
    )
    ((eql str-line 'eof) (close str) (reverse ammo-list))
  )
)

```

```
;;-----  
;;BUILD-WEIGHTS  
;;Opens and reads in the weight values  
;;-----  
(defun build-weights (data-path)  
  (setf weight-path (make-pathname :name data-path))  
  (setf weight-str (open weight-path :direction :input  
                        :if-exists :supersede))  
  (setf my-weights1 (read-line weight-str nil))  
  (close weight-str)  
)
```


APPENDIX F: INPUT DATA

4.2	5.1	6.0	7.0	8.4	10.3	12.4	14.9	17.5	21.0	24.8	28.9	30.9	32.6	32.9	31.6	28.8	25.0	21.4	17.5	14.8	11.8	9.8	7.6	6.6	5.2	4.8	4.1	3.9	2.8	2.2
4.2	5.3	6.3	7.4	8.9	11.0	13.1	15.9	18.9	23.0	27.3	31.6	33.7	35.3	35.2	33.7	31.1	26.8	22.6	18.2	15.2	12.1	10.0	7.9	6.7	5.3	4.9	4.1	3.9	2.8	2.2
4.3	5.5	6.5	7.6	9.1	11.1	13.5	16.5	19.7	24.0	28.4	32.6	34.7	35.9	35.9	34.3	31.5	27.0	22.0	18.0	14.9	12.0	9.8	7.9	6.7	5.3	4.8	4.1	3.9	2.7	2.2
4.3	5.3	6.5	7.4	9.0	10.9	13.4	16.2	19.5	23.4	27.9	32.0	34.1	34.7	34.6	32.5	28.4	24.0	20.0	16.0	13.7	11.1	9.1	7.4	6.5	5.2	4.6	3.9	3.6	2.5	2.2
4.2	5.3	6.2	7.3	8.8	10.4	12.8	15.4	18.3	21.7	25.7	28.7	30.6	29.9	29.6	26.7	21.9	17.8	15.4	12.7	10.8	9.3	8.0	6.6	5.9	4.8	4.3	3.8	3.5	2.5	2.2
4.2	5.0	5.9	6.9	8.2	9.5	11.4	13.5	15.8	18.1	20.7	21.9	22.7	19.7	19.0	13.5	11.5	9.4	9.0	8.2	8.0	7.2	6.5	5.7	5.0	4.3	4	3.5	3.3	2.3	1.9
3.9	4.8	5.5	6.2	7.3	8.3	9.6	11.3	12.4	13.2	14.5	12.8	11.8	6.5	6.0	0.4	-0.5	-0.7	2.2	3.1	4.1	4.5	4.9	4.3	4.3	3.9	3.6	3.3	3.1	2.1	1.9
3.8	4.3	4.9	5.5	6.5	6.9	7.7	8.0	8.3	7.3	6.7	0.9	-0.7	-8.6	-8.3	-14.7	-12.1	-9.7	-12.9	-2.9	0.5	1.4	2.8	3.1	3.3	3.2	3.2	2.9	2.8	1.9	1.9
3.5	3.9	4.3	4.7	5.2	5.2	5.7	4.8	3.8	1.1	-1.9	-10.2	-14.6	-23.8	-24.7	-28.2	-23	-19.4	-12.4	-8.3	-3.5	-1.4	0.8	1.5	2.4	3.5	2.6	2.5	2.6	2.3	1.9
3.2	3.5	3.6	3.9	4.1	3.5	3.3	1.2	-0.5	-5.2	-11.4	-21.3	-30.4	-35.4	-36	-36	-32.8	-27.8	-19.6	-13	-7.2	-4.2	-1.1	0.4	1.4	1.8	2.2	2.2	2.4	2.1	1.8
2.9	3.1	3.2	3.2	3.1	2.1	1.5	-1.6	-4.2	-10.7	-17.9	-29.6	-36	-36	-36	-36	-34	-25.8	-17.9	-11	-6.6	-2.9	-0.9	0.5	1.2	1.6	1.9	2.1	1.9	1.9	1.6
2.6	2.6	2.6	2.5	2.1	1.0	0.4	-3.3	-6.2	-13.8	-20.7	-33.9	-36	-36	-36	-36	-36	-30	-20.9	-13.9	-8.3	-4.3	-1.9	-0.5	0.8	1.4	1.6	1.9	1.9	1.6	1.6
2.5	2.4	2.2	1.9	1.4	0.4	-1.0	-4.6	-7.7	-15.9	-23.8	-35.6	-36	-36	-36	-36	-36	-22	-22.3	-15.2	-9.3	-5.2	-2.5	-0.7	0.4	1.1	1.5	1.8	1.8	1.5	1.5
2.2	2.1	1.9	1.5	0.9	-0.5	-1.6	-5.5	-8.7	-16.4	-24.2	-35	-36	-36	-36	-36	-36	-32.2	-22.3	-15.5	-9.3	-5.5	-2.8	-0.9	0.4	0.9	1.2	1.6	1.5	1.3	1.3
2.1	1.9	1.8	1.4	0.8	-0.6	-1.8	-5.3	-8.0	-15.1	-21.9	-32	-36	-36	-36	-36	-36	-35.6	-29.8	-20.3	-14.7	-9	-5.3	-2.8	-0.9	0.4	0.8	1.2	1.5	1.5	1.3
2.0	1.8	1.6	1.2	0.7	-0.6	-1.6	-4.4	-7.0	-12.4	-17.2	-27.1	-32.9	-36	-36	-36	-36	-32.6	-24.8	-17.5	-13.1	-8.2	-4.9	-2.6	-0.9	0.4	0.7	2.1	1.5	1.5	1.3
1.9	1.6	1.5	1.2	0.7	-0.5	-1.4	-3.6	-5.7	-9.8	-13.8	-21	-24.8	-31.2	-32.2	-33.3	-30.1	-24.8	-19.3	-14.4	-10	-6.6	-3.9	-2.1	-0.7	0.4	0.8	1.1	1.4	1.4	1.3
1.8	1.6	1.5	1.1	0.7	-0.5	-1.1	-2.8	-4.5	-7.3	-10.2	-15.4	-17.3	-21.9	-23.0	-24.8	-20.5	-16.8	-13.9	-10	-7.6	-4.9	-3.1	-1.5	-0.5	0.4	0.8	1.1	1.4	1.4	1.3
1.6	1.5	1.1	0.8	0.4	-0.7	-1.8	-3.1	-5	-7.3	-10.1	-12.4	-15.1	-16.1	-17.0	-15.1	-12.5	-10.6	-7.9	-5.9	-3.9	-2.2	-1.1	-0.5	0.4	0.8	1.1	1.4	1.4	1.3	1.3
1.6	1.5	1.4	1.1	0.8	0.4	-0.5	-1.5	-2.4	-3.9	-5.5	-7.7	-9.7	-11.7	-12.7	-13.0	-12	-10	-8	-6	-4.3	-2.8	-1.6	-0.7	0.4	0.5	0.9	1.1	1.4	1.4	1.3
1.6	1.5	1.2	1.1	0.8	0.4	-0.5	-1.2	-2.1	-3.5	-4.9	-7.3	-9	-11.4	-12.4	-12.5	-11.5	-9.8	-7.6	-5.5	-3.9	-2.6	-1.3	-0.7	0.4	0.5	0.9	1.1	1.3	1.5	1.2
1.5	1.4	1.2	0.9	0.7	0.4	-0.5	-1.4	-2.2	-3.9	-5.3	-8.4	-10.3	-13.4	-14.5	-14.9	-13.5	-11.8	-8.9	-6	-4.3	-2.9	-1.5	-0.7	-0.4	0.5	0.8	0.9	1.2	1.5	1.2
1.5	1.2	1.1	0.9	0.5	-0.4	-0.7	-1.8	-2.6	-4.8	-6.5	-10.3	-12.5	-16.4	-17.9	-18.5	-16.9	-14.7	-11.1	-7.7	-5.3	-3.6	-1.9	-0.9	-0.5	0.4	0.7	0.9	1.1	1.3	1.1
1.4	1.1	0.9	0.7	0.4	-0.5	-0.9	-2.4	-3.3	-5.7	-7.9	-12.5	-15.8	-20.2	-22.2	-23	-20.7	-18.1	-13.7	-9.8	-6.9	-4.6	-2.8	-1.4	-0.7	0.4	0.6	0.8	1.1	1.2	1.1
1.2	1.1	0.9	0.5	0.4	-0.7	-1.4	-2.8	-4.1	-6.7	-9.4	-14.4	-18.2	-23	-25.7	-26.5	-24.4	-21	-16.1	-11.8	-8.2	-5.6	-3.3	-1.9	-0.8	0.4	0.4	0.7	0.9	1.1	1.1
1.2	0.9	0.8	0.4	-0.5	-0.8	-1.6	-3.2	-4.8	-7.4	-10.4	-15.6	-19.7	-25	-27.8	-28.8	-26.7	-23.1	-17.9	-13	-9.3	-6.6	-3.4	-2.4	-1.1	-0.5	0.4	0.5	0.9	1.1	1.1
1.1	0.9	0.7	0.4	-0.5	-0.9	-1.9	-3.3	-5	-7.9	-11	-16.1	-20.5	-25.5	-28.7	-29.5	-27.7	-24.1	-18.9	-13.9	-10	-7	-4.5	-2.6	-1.5	-0.7	-0.5	0.4	0.8	1.1	1.1
1.1	0.9	0.7	0.4	-0.5	-0.9	-2.1	-3.4	-5.2	-7.9	-11	-15.5	-20	-24.4	-27.7	-28.5	-27.1	-23.4	-18.8	-14	-10	-7.3	-4.6	-2.9	-1.6	-0.7	-0.5	0.4	0.8	1.1	1.1
1.1	0.8	0.5	0.4	-0.5	-0.9	-2.1	-3.3	-5	-7.4	-10.3	-14.2	-18.5	-22	-25.3	-26	-24.8	-21.4	-17.3	-13.4	-9.6	-6.9	-4.9	-2.9	-1.6	-0.8	-0.7	0.4	0.7	0.9	0.9
0.9	0.8	0.5	0.4	-0.5	-0.9	-1.9	-3.1	-4.6	-6.7	-9.4	-12.5	-16	-19.2	-21.7	-22	-20.9	-18.3	-14.9	-11.8	-8.4	-6.2	-4.3	-2.6	-1.6	-0.7	-0.7	0.4	0.7	0.9	0.9
0.9	0.8	0.5	0.4	-0.5	-0.8	-1.8	-2.6	-4.2	-5.9	-8.3	-10.7	-13.5	-15.6	-18.2	-17.9	-17.5	-14.9	-12.8	-10	-7.3	-5.5	-3.9	-2.4	-1.5	-0.7	-0.6	0.4	0.7	0.9	0.9
0.9	0.8	0.5	0.4	-0.5	-0.7	-1.5	-2.2	-3.5	-4.9	-6.7	-8.6	-10.7	-14.4	-14.1	-13.8	-13.7	-11.8	-10.1	-8	-6.2	-4.5	-3.3	-2.1	-1.2	-0.6	-0.5	0.4	0.7	0.9	0.9
0.9	0.8	0.5	0.4	-0.4	-0.6	-1.1	-1.8	-2.8	-3.9	-5.3	-6.7	-8.2	-9.6	-10.7	-10.6	-10.1	-9.1	-7.9	-6.5	-4.8	-3.6	-2.8	-1.5	-0.9	-0.5	-0.5	0.4	0.8	0.9	0.9
0.9	0.9	0.7	0.4	0.4	-0.5	-0.8	-1.3	-2.1	-2.9	-3.9	-4.9	-6	-7.2	-7.6	-7.6	-7.6	-6.7	-5.9	-4.6	-3.6	-2.8	-2.1	-1.1	-0.8	-0.5	-0.5	0.5	0.8	0.9	0.9
0.9	0.9	0.8	0.5	0.4	-0.5	-0.6	-0.8	-1.4	-2.1	-2.8	-3.6	-4.2	-5	-5.5	-5.3	-5.2	-4.6	-4.2	-3.2	-2.5	-1.9	-1.5	-0.7	-0.7	0.4	0.4	0.5	0.8	0.9	0.9

Training Set 1: 3-5in Rocket

1.5	1.6	1.8	2	2.1	2.4	2.6	2.8	3.2	3.3	3.5	3.8	3.9	4.1	4.1	3.9	3.9	3.8	3.6	3.3	2.9	2.6	2.5	2.2	2.1	1.9	1.8	1.6	1.5	1.5	1.4
1.6	1.8	1.9	2.1	2.4	2.6	2.9	3.2	3.6	3.9	4.1	4.5	4.8	4.8	4.8	4.9	4.6	4.3	4.2	3.9	3.3	3.1	2.8	2.5	2.2	2.1	1.9	1.6	1.6	1.5	1.5
1.8	1.9	2.1	2.4	2.6	3.1	3.3	3.8	4.3	4.6	4.9	5.5	5.9	5.9	6	6	5.6	5.3	4.9	4.5	3.9	3.5	3.2	2.8	2.5	2.2	2.1	1.8	1.6	1.6	1.5
1.9	2.1	2.3	2.5	2.8	3.3	3.8	4.3	4.9	5.5	5.7	6.7	7.2	7.4	7.2	7.3	6.7	6.5	5.7	5.3	4.5	3.9	3.5	3.1	2.8	2.4	2.2	1.9	1.8	1.6	1.5
1.9	2.1	2.5	2.8	3.2	3.6	4.2	4.9	5.7	6.3	6.9	8.2	8.3	9.1	9	9	8.4	7.7	6.9	6.2	5.2	4.5	3.8	3.3	2.9	2.6	2.2	2.1	1.8	1.6	1.5
1.9	2.2	2.6	2.9	3.3	3.9	4.6	5.5	6.7	7.4	8.4	10	10.8	11.4	11.3	11.3	10.3	9.4	7.9	7.2	5.7	4.9	4.2	3.5	3.1	2.6	2.4	2.1	1.9	1.8	1.6
2.1	2.4	2.6	3.1	3.5	4.2	5	6	7.6	8.7	10.1	12.1	13	13.9	13.8	13.8	12.5	11.1	9.4	8	6.7	5.5	4.6	3.8	3.2	2.8	2.5	2.2	1.9	1.8	1.6
2.1	2.5	2.8	3.2	3.8	4.5	5.5	6.6	8.4	9.7	11.4	14.1	15.1	16.4	16.2	15.9	14.4	12.7	10.3	8.9	7.2	5.7	4.9	3.9	3.3	2.9	2.6	2.2	2.1	1.8	1.6
2.1	2.5	2.8	3.2	3.8	4.6	5.6	6.7	8.9	10.6	12.5	15.2	16.8	18.2	18.2	17.3	15.8	13.5	11.1	9.3	7.9	6	4.9	4.1	3.5	2.9	2.6	2.4	2.1	1.9	1.6
2.1	2.5	2.8	3.2	3.8	4.5	5.6	6.7	8.9	10.6	12.5	14.8	16.6	17.5	17.9	16.4	15.5	13	10.8	9	7.7	5.9	4.9	3.9	3.3	2.9	2.5	2.2	2.1	1.8	1.6
2.1	2.4	2.6	3.1	3.5	4.3	5	6.2	7.9	9.1	10.7	12	13	12.7	13.2	11.7	12.5	10.6	9.4	8	6.7	5.5	4.6	3.8	3.3	2.8	2.5	2.2	2.1	1.8	1.6
1.9	2.3	2.5	2.8	3.3	3.9	4.5	5.3	6.3	6.9	7.7	6.9	6.7	4.5	2.9	3.3	6.5	5.6	6.6	6	5.5	4.6	3.9	3.5	3.1	2.6	2.4	2.2	1.9	1.8	1.6
1.9	2.2	2.4	2.6	2.8	3.2	3.6	3.9	4.5	3.9	3.6	-0.7	-2.4	-7.2	-8.9	-8	-2.4	-0.7	2.4	2.9	3.6	3.5	3.3	2.9	2.6	2.4	2.2	2	1.8	1.6	1.6
1.8	2	2.1	2.2	2.4	2.5	2.6	2.2	1.5	0.7	-1.2	-7.7	-12.8	-19.9	-21.4	-20.2	-12.7	-7.3	-1.9	0.4	1.6	2.1	2.4	2.4	2.2	2.1	2.1	1.9	1.8	1.6	1.5
1.6	1.8	1.9	1.9	1.8	1.8	1.6	0.8	-0.9	-3.3	-6.2	-14.8	-20.6	-29.2	-30.9	-29.5	-21.3	-13.9	-8	-3.3	-0.7	0.5	1.1	1.6	1.8	1.8	1.8	1.6	1.6	1.6	1.5
1.5	1.6	1.6	1.6	1.4	1.1	0.4	-0.7	-3.1	-6.7	-10	-19.9	-27	-34	-35.7	-34.5	-27.4	-18.8	-11.7	-6.2	-2.6	-0.7	0.4	0.9	1.2	1.5	1.5	1.5	1.5	1.5	1.4
1.4	1.4	1.4	1.3	0.9	0.4	-0.5	-1.8	-4.6	-8.5	-12.4	-20.8	-30.5	-36	-36	-36	-30.6	-21.4	-14.1	-8	-4.3	-1.6	-0.7	0.5	0.9	1.2	1.2	1.4	1.4	1.4	1.4
1.2	1.2	1.1	0.9	0.5	0.4	-0.9	-2.4	-5.6	-10	-14.3	-22.6	-31.2	-35.7	-36	-35.6	-30.9	-22.2	-15.1	-9.3	-5	-2.4	-1.1	0.4	0.5	0.9	1.1	1.2	1.2	1.4	1.2
1.2	1.1	0.9	0.9	.4	-0.5	-1.2	-2.6	-5.5	-10	-14.5	-20	-28.8	-33.6	-35.4	-33.9	-30.4	-21.9	-15.1	-9.8	-5.5	-3.1	-1.6	-0.5	0.4	0.7	0.9	1.1	1.2	1.2	1.2
1.1	1.1	0.9	0.8	0.4	-0.5	-1.4	-2.6	-5.2	-9.1	-13	-17.5	-24.3	-29.2	-32	-29.5	-26.1	-19	-13.4	-9.1	-5.5	-3.1	-1.8	-0.5	0.4	0.5	0.8	0.9	1.1	1.2	1.2
1.1	0.9	0.8	0.6	0.4	-0.5	-1.2	-2.4	-4.5	-7.2	-10.8	-14.5	-18.8	-22.4	-26	-22.7	-21.2	-15.8	-11.3	-7.9	-4.9	-2.8	-1.6	-0.5	0.4	0.5	0.8	0.9	1.1	1.2	1.2
0.9	0.9	0.8	0.5	0.4	-0.5	-0.9	-2.1	-3.9	-6.1	-8.4	-10.7	-13.9	-16.6	-17.9	-16.4	-15.8	-12.3	-8.9	-6.5	-4.2	-2.4	-1.5	-0.5	0.4	0.5	0.7	0.8	0.9	1.1	1.1
0.9	0.9	0.8	0.6	0.4	-0.5	-0.8	-1.5	-2.9	-4.6	-6.5	-7.9	-9.8	-11.7	-12.8	-11.5	-11	-8.9	-6.7	-5	-3.1	-1.8	-1.1	-0.5	0.4	0.5	0.7	0.8	0.9	1.1	1.1
0.9	0.9	0.8	0.7	0.4	-0.4	-0.7	-1.1	-2.1	-3.3	-4.3	-5.3	-6.7	-7.9	-8	-7.7	-7.4	-6.5	-4.8	-3.8	-2.2	-1.2	-0.8	-0.4	0.4	0.5	0.7	0.8	0.9	1.1	1.1
0.9	0.9	0.8	0.7	0.4	0.4	-0.5	-0.7	-1.4	-2.2	-3.1	-3.8	-4.5	-4.9	-5.2	-4.9	-4.9	-4.2	-3.3	-2.4	-1.5	-0.8	-0.5	0.4	0.4	0.5	0.7	0.8	0.9	1.1	1.1
0.9	0.9	0.8	0.8	0.5	0.4	0.4	-0.5	-0.8	-1.4	-1.9	-2.2	-2.9	-3.2	-3.5	-3.2	-3.2	-2.5	-1.9	-1.4	-0.9	-0.5	-0.5	0.4	0.4	0.7	0.8	0.8	0.9	1.1	1.1
0.9	0.9	0.9	0.8	0.7	0.4	0.4	-0.4	-0.5	-0.7	-1.1	-1.4	-1.8	-1.9	-1.9	-1.9	-1.6	-1.2	-0.8	-0.5	-0.4	0.4	0.4	0.5	0.8	0.8	0.9	0.9	1.1	1.1	1.1
1.1	0.9	0.9	0.9	0.8	0.7	0.4	0.4	0.4	-0.5	-0.5	-0.7	-0.9	-0.9	-1.1	-0.9	-0.1	-0.7	-0.7	-5	-0.4	0.4	0.4	0.5	0.7	0.8	0.9	0.9	0.9	1.1	1.1
1.1	1	0.9	0.9	0.8	0.7	0.7	0.4	0.4	0.4	-0.4	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	0.4	0.4	0.4	0.5	0.7	0.8	0.8	0.4	0.9	0.9	1.1	1.1
1.1	1.1	1	0.9	0.9	0.8	0.8	0.7	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.7	0.7	0.8	0.9	0.9	0.9	0.9	1.1	1.1	1.1
1.1	1.1	1.1	0.9	0.9	0.9	0.9	0.8	0.7	0.7	0.5	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.7	0.8	0.8	0.8	0.8	0.9	0.9	0.9	0.9	1.1	1.1	1.1
1.1	1.1	1.1	1	0.9	0.9	0.9	0.9	0.8	0.8	0.8	0.8	0.5	0.7	0.5	0.7	0.7	0.7	0.8	0.8	0.9	0.9	0.9	0.9	0.9	0.9	0.9	1.1	1.1	1.1	1.1
1.1	1.1	1.1	1.1	1.1	1.1	1.1	0.9	0.9	0.9	0.9	0.8	0.8	0.8	0.8	0.8	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	1.1	1.1	1.1	1.1	1.1
1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	0.9	0.9	1.1	0.9	0.9	0.9	0.9	0.9	0.9	1.1	0.9	0.9	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1
1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	0.9	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.2	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1	1.1

Training Set 1: 60mm Mortar

3.3	3.8	4.3	4.9	5.6	6.5	7.4	8.7	10	11.5	12.8	13.9	15.2	16.2	15.6	15.9	14.2	13.2	12	10.7	9.7	8.2	7.2	6.2	5.5	4.6	4	3.6	3.2	2.9	2.6		
3.5	4.1	4.6	5.5	6.2	7.2	8.4	9.7	11.7	13.2	15.1	16.9	19	19.5	19.3	19.5	17.9	16.2	14.5	13	11.3	9.4	8	6.9	6.2	5.2	4.3	3.9	3.5	3.1	2.8		
3.8	4.3	5	5.9	6.7	7.9	9.6	11	13.7	15.8	18.6	21.2	23	24.3	24	24.3	21.7	19.6	17.2	14.9	13	10.8	8.9	7.4	6.7	5.5	4.8	4.2	3.6	3.2	2.9		
3.9	4.6	5.3	6.5	7.3	8.7	10.4	12.4	15.2	17.9	21.4	24.1	27.1	28.5	28.4	28.4	25.5	22.6	19.7	16.8	14.7	11.8	9.7	8	7.2	5.9	5	4.3	3.8	3.3	3.1		
4.2	4.8	5.6	6.7	7.9	9.4	11.3	13.5	16.9	19.6	23.7	27.8	30.8	31.8	32.5	31.6	28.9	25.5	22	18.5	16.1	12.8	10.4	8.6	7.6	6.2	5.2	4.5	3.9	3.5	3.1		
4.2	4.9	5.7	6.9	8.2	9.8	11.8	14.1	17.8	21.2	25.4	29.5	32.6	34	34.3	33.6	30.9	27.7	23.8	19.7	16.9	13.9	11.1	9	7.9	6.5	5.5	4.6	4.1	3.5	3.2		
4.3	5	5.9	7	8.4	10	12	14.4	18.1	21.4	25.8	29.8	32.8	34.2	34.5	33.5	31.2	28.2	24.3	20.2	17.1	14.4	11.3	9.3	8	6.6	5.5	4.8	4.1	3.6	3.2		
4.3	5	5.9	7	8.3	9.8	12	14.1	17.5	20.7	24.3	28.1	30.6	31.9	31.8	30.5	29.4	27	23.4	19.7	16.8	14.1	11.1	9.1	8	6.6	5.6	4.8	4.1	3.6	3.2		
4.2	4.9	5.7	6.7	8	9.4	11.3	13.2	16.1	18.6	21	23.4	24.7	26.4	24.8	24.7	23.6	23.1	20.3	17.9	15.6	12.8	10.8	8.9	7.9	6.5	5.5	4.6	4.1	3.6	3.2		
4.2	4.8	5.5	6.5	7.6	8.9	10.4	11.7	13.9	15.8	16.2	15.9	15.2	14.8	13.5	14.5	16.2	17.6	17.1	15.2	13.9	11.5	9.8	8.3	7.4	6	5.3	4.5	3.9	3.5	3.2		
3.9	4.5	5	6.6	7	7.9	8.9	10	10.8	11.5	11.7	8	2.6	1.5	-1.1	1.4	5.7	9.6	11.7	11.7	11.4	9.7	8.7	7.4	6.9	5.6	4.9	4.3	3.9	3.3	3.2		
3.8	4.2	4.6	5.3	6.2	6.7	7.2	7.6	7.7	7	3.3	-1.2	-8.4	-12.7	-14.2	-12	-5.3	0.4	5.2	6.7	8.4	7.6	7.2	6.2	6	5	4.5	4.1	3.6	3.2	3.1		
3.5	3.9	4.2	4.6	5	5.5	5.3	5.3	3.6	1.8	-4.8	-11.3	-19.2	-26	-29.8	-26.1	-16.8	-7.9	-1.5	7.9	3.9	5.5	5.6	5.2	5	4.5	4.2	3.6	3.3	3.2	2.9		
3.2	3.3	3.6	3.9	4.1	3.8	3.5	2.6	-0.4	-2.6	-10.4	-18.9	-31.9	-35.9	-36	-35.6	-27.8	-17.5	-7.6	-3.5	0.4	2.1	3.5	3.8	3.9	3.8	3.6	3.3	3.2	2.8	2.8		
2.9	2.9	3.1	3.1	2.9	2.4	1.6	-0.4	-3.4	-8.9	-16.2	-27.8	-36	-36	-36	-36	-35.3	-25.4	-15.5	-8.4	-3.8	-0.5	1.1	2.2	2.9	2.9	3.1	2.9	2.8	2.6	2.6		
2.5	2.6	2.4	2.2	2.1	1.1	-0.4	-2.8	-7.6	-12.9	-23.4	-33.2	-36	-36	-36	-36	-36	-32	-21.2	-13.4	-7.4	-2.8	-0.7	9	1.8	2.2	2.4	2.5	2.5	2.4	2.4		
2.4	2.2	1.9	1.6	1.1	-0.4	-1.9	-4.8	-10.7	-15.2	-27.7	-36	-36	-36	-36	-36	-36	-35.6	-25.7	-16.9	-10.7	-5	-2.4	-0.5	0.7	1.6	1.9	2.1	2.2	2.2	2.2		
2.1	1.8	1.4	1.1	0.4	-1.1	-3.8	-6.7	-12.4	-19.2	-30.9	-36	-36	-36	-36	-36	-36	-36	-29.6	-20.2	-13.2	-7	-3.8	-1.2	-0.5	0.9	1.4	1.8	1.9	2.1	2.1		
1.8	1.5	1.1	1.4	-0.5	-1.9	-4.6	-8.3	-13.8	-19	-31.9	-36	-36	-36	-36	-36	-36	-36	-32.5	-23	-15.1	-8.7	-5.2	-2.4	-0.9	0.4	0.9	1.4	1.6	1.8	1.9		
1.6	1.2	0.8	0.4	-1.1	-2.5	-5	-8.6	-13.9	-20.5	-32	-36	-36	-36	-36	-36	-36	-36	-32.9	-23.6	-16.8	-9.4	-6	-3.1	-1.4	-0.5	0.5	1.1	1.5	1.6	1.6		
1.5	0.9	0.5	-0.5	-1.2	-2.9	-5.5	-8.9	-14.2	-20.2	-31.1	-36	-36	-36	-36	-36	-36	-36	-32.6	-24.8	-17.2	-10	-6.4	-3.5	-1.8	-0.5	0.4	0.9	1.2	1.5	1.6		
1.2	0.9	0.4	-0.5	-1.4	-2.9	-5.5	-8.4	-13.5	-18.8	-28.4	-36	-36	-36	-36	-36	-36	-36	-30.9	-22.7	-16.2	-10	-6.4	-3.6	-2.1	-0.7	0.4	0.8	1.1	1.4	1.5		
1.2	0.8	0.4	0.5	-1.4	-2.8	-5	-7.9	-12.4	-16.9	-25.1	-33.3	-36	-36	-36	-36	-36	-36	-34.7	-28.1	-20.3	-14.7	-9.4	-6	-3.5	-2.1	-0.7	0.4	0.7	0.9	1.2	1.4	
1.1	0.8	0.4	0.5	-1.2	-2.5	-4.3	-6.7	-10.7	-13.9	-20.6	-27.4	-33.9	-36	-36	-36	-36	-36	-34.6	-29.5	-23	-17.2	-12.5	-8.3	-5.5	-3.2	-1.8	-0.7	0	0.7	0.9	1.2	1.4
1.1	0.8	0.4	0.5	-1.1	-2.2	-3.8	-5.7	-8.9	-11.7	-16.2	-21.6	-26	-31.1	-30.9	-31.1	-26.1	-22.6	-18.2	-13.8	-10.3	-6.9	-4.5	-2.5	-1.5	-0.5	0.4	0.7	0.9	1.1	1.4		
1.1	0.8	0.4	-0.4	-0.8	-1.6	-3.2	-4.6	-7.3	-9.1	-12.5	-16.4	-20.3	-22.3	-24.3	-21.4	-19.9	-17.6	-13.9	-11	-8.4	-5.6	-3.8	-2.1	-1.2	-0.5	0.4	0.7	0.9	1.1	1.4		
1.1	0.8	0.5	0.4	-0.7	-1.2	-2.4	-3.5	-5.3	-6.9	-9.3	-12.3	-14.2	-15.8	-17.2	-15.4	-14.5	-13	-10.6	-8.2	-6.3	-4.3	-2.8	-1.5	-0.8	-0.5	0.4	0.8	0.9	1.2	1.2		
1.2	0.9	0.7	0.4	-0.5	-0.7	-1.6	-2.5	-3.4	-4.9	-6.6	-8.6	-10.4	-11.1	-11.5	-10.6	-10	-9.4	-7.7	-6	-4.5	-3.1	-1.9	-0.9	-0.5	0.4	0.5	0.9	0.9	1.2	1.2		
1.2	0.9	0.8	0.5	0.4	-0.5	-0.9	-1.6	-2.6	-3.2	-4.6	-5.9	-6.7	-7.7	-7.6	-7.2	-6.9	-6.5	-5.3	-4.2	-3.2	-2.1	-1.1	-0.7	-0.5	0.4	0.7	0.9	1.1	1.2	1.2		
1.2	1.1	0.9	0.7	0.4	-0.4	-0.5	-0.9	-1.6	-2.2	-3.1	-3.9	-4.3	-5.3	-4.8	-4.9	-4.5	-4.1	-3.3	-2.8	-2.1	-1.4	-0.8	-0.5	0.4	0.4	0.8	0.9	1.1	1.2	1.4		
1.2	1.1	1.1	0.8	0.5	0.4	-0.5	-0.5	-0.9	-1.2	-1.8	-2.5	-2.8	-3.2	-3.2	-3.1	-2.9	-2.6	-2.1	-1.6	-1.1	-0.7	-0.5	0.4	0.4	0.7	0.9	1.1	1.2	1.4	1.4		
1.4	1.2	1.1	0.9	0.8	0.5	0.5	0.4	-0.5	-0.7	-0.9	-1.2	-1.6	-1.8	-1.8	-1.6	-1.6	-1.4	-1.1	-0.7	-0.7	-0.5	0.4	0.5	0.7	0.8	1.1	1.2	1.2	1.4	1.4		
1.5	1.2	1.2	1.1	0.9	0.8	0.5	0.4	0.4	-0.5	-0.5	-0.7	-0.7	-0.8	-0.8	-0.8	-0.7	-0.7	-0.5	-0.5	0.4	0.4	0.7	0.9	0.9	1.2	1.2	1.4	1.4	1.4	1.4		
1.5	1.4	1.2	1.2	1.1	1.1	0.8	0.7	0.5	0.4	0.4	-0.4	-0.4	-0.5	-0.4	-0.4	-0.4	0.4	0.4	0.4	0.5	0.8	0.9	1.1	1.1	1.2	1.2	1.4	1.4	1.4	1.4		
1.5	1.4	1.4	1.2	1.2	1.2	1.1	0.9	0.8	0.7	0.7	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.4	0.5	0.8	0.9	1.1	1.1	1.1	1.4	1.4	1.4	1.4	1.4	1.4		

Training Set 1: 81mm Mortar

2.4	2.6	2.6	2.8	2.8	3.1	3.2	3.2	3.3	3.2	3.2	3.1	3	2.9	2.8	2.7	2.6	2.6	2.6	2.6	2.6	2.6	2.5	2.5	2.4	2.5	2.2	2.2	2.1	2.1
2.6	2.8	2.8	3.1	3.1	3.2	3.3	3.3	3.5	3.3	3.3	3.2	3.1	3	2.8	2.6	2.6	2.6	2.6	2.6	2.6	2.6	2.6	2.5	2.5	2.4	2.4	2.2	2.1	
2.8	3.1	3.1	3.3	3.4	3.6	3.8	3.8	3.8	3.7	3.6	3.2	3.1	2.9	2.6	2.6	2.5	2.6	2.6	2.6	2.8	2.8	2.8	2.8	2.6	2.6	2.5	2.4	2.2	2.2
3.1	3.3	3.3	3.6	3.8	3.9	4.1	4.1	4.1	3.9	3.7	3.2	2.9	2.6	2.2	2.2	2.2	2.2	2.4	2.5	2.6	2.8	2.8	2.9	2.9	2.8	2.8	2.6	2.6	2.4
3.3	3.5	3.6	3.9	4.2	4.3	4.5	4.3	4.3	4.1	3.8	3.1	2.5	1.9	1.4	1.5	1.4	1.6	1.9	2.4	2.6	2.8	2.9	3.1	3.1	3	2.9	2.8	2.6	2.4
3.6	3.9	3.9	4.3	4.5	4.8	4.9	4.9	4.7	4.3	3.6	2.6	1.6	0.7	0.4	0.4	0.4	0.7	1.2	2	2.4	2.8	3.1	3.2	3.2	3.2	3.1	2.9	2.8	2.6
3.9	4.2	4.3	4.9	5	5.3	5	5.3	5.1	4.5	3.5	2.1	0.4	-0.7	-2.2	-1.9	-1.6	-0.7	0.4	1.5	2.1	2.6	3.1	3.3	3.3	3.3	3.2	3.1	2.8	2.6
4.2	4.5	4.7	5.3	5.6	6	6.2	6	5.6	4.6	3.2	0.9	-1.5	-3.3	-5.5	-4.9	-4.5	-2.8	-0.9	0.9	1.8	2.6	3.2	3.3	3.6	3.5	3.5	3.3	3.2	3.1
4.5	4.9	5	5.9	6.3	6.7	7	6.7	6.3	5	3.1	-0.5	-4	-6.9	-9.4	-8.7	-7.4	-5.3	-2.4	0.4	1.5	2.6	3.3	3.6	3.9	3.9	3.8	3.5	3.4	3.2
4.9	5.3	5.6	6.5	7	7.6	7.9	7.9	7.3	5.7	3.2	-0.9	-6	-9.8	-13.3	-12.4	-10.3	-7.3	-3.5	-0.5	1.4	2.6	3.5	3.9	4.1	4.2	4	3.8	3.6	3.3
5.2	5.7	6	7.1	7.9	8.5	9.1	9.1	8.7	7.1	4.1	-0.7	-7.3	-12.3	-16.2	-15.5	-13.1	-9.4	-4.6	-0.7	1.2	2.8	3.9	4.2	4.5	4.3	4.3	4.1	3.8	3.5
5.5	6.2	6.5	7.7	8.6	9.4	10.1	10.6	10.4	9	5.9	0.7	-6.6	-13.1	-16.8	-17.1	-14.2	-10.3	-4.9	-0.7	1.6	3.2	4.3	4.6	4.9	4.8	4.5	4.3	4	3.8
5.6	6.6	6.8	8.3	9.3	10.3	11.4	12.1	12.5	11.4	8.7	3.8	-3.8	-10.8	-14.2	-15.9	-13.5	-10.1	-4.5	-0.4	2.2	3.8	4.9	5.2	5.3	5	4.9	4.5	4.2	3.9
5.9	6.9	7.2	8.7	9.8	11.1	12.5	13.5	14.5	13.9	12.5	8.2	2.2	-5.3	-8.4	-12.6	-10.8	-8.3	-3.1	0.9	3.2	4.5	5.5	5.6	5.6	5.5	5.2	4.8	4.5	4.1
6.2	7	7.4	9	10.4	11.7	13.3	14.8	15.8	16.1	15.2	12.8	8.6	1.9	-1.1	-6.9	-6.9	-5	-1.2	2.2	4.2	5.3	6.2	6.2	6.2	5.7	5.5	5	4.6	4.2
6.2	7.2	7.6	9.2	10.6	12	13.5	15.1	16.2	16.8	16.6	15.2	12.4	8.7	4.3	-1.4	-2.8	-2.2	0.4	2.9	4.9	6.2	6.6	6.6	6.5	6	5.6	5.2	4.8	4.3
6.2	7.2	7.7	9.2	10.6	11.8	13.3	14.8	15.5	15.7	15.5	13.5	11.5	9.3	4.1	0.5	-1.5	-1.2	0.9	3.6	5.3	6.3	6.7	6.7	6.6	6.2	5.7	5.3	4.9	4.4
6.2	7.2	7.6	9.1	10.3	11.4	12.5	13.5	13.4	13	11.2	9.6	4.5	2.7	-5	-5.5	-5.7	-3.5	-0.7	2.8	4.8	6	6.6	6.7	6.6	6.3	5.7	5.3	4.9	4.5
6.2	7	7.4	8.9	9.8	10.7	11.5	11.8	11.1	9	6	0.4	-7.4	-13.1	-22.3	-19.2	-16.9	-9.8	-5	0.4	3.2	5.2	6	6.5	6.3	6.2	5.6	5.2	4.9	4.5
6	6.9	7.3	8.4	9.3	10	10.1	9.9	8.3	5.5	-0.9	-9.7	-22.7	-30.6	-36	-33.2	-28.8	-19.8	-10.9	-3.1	1.1	3.9	5.2	5.7	6	5.9	5.5	5.2	4.8	4.3
5.9	6.6	7	7.9	8.6	9	8.9	8.1	5.6	1.5	-6.6	-18.5	-33.7	-36	-36	-36	-36	-28.8	-16.5	-6.9	-0.9	2.2	3.9	5.2	5.5	5.5	5.3	4.9	4.6	4.3
5.6	6.3	6.6	7.4	7.9	8.2	7.9	6.5	3.9	-1.2	-9.6	-23.4	-36	-36	-36	-36	-36	-33.5	-20.2	-9.3	-2.6	1.1	3.2	4.3	4.8	5	4.9	4.6	4.3	4.2
5.5	6	6.3	6.9	7.2	7.3	6.9	5.3	2.6	-2.2	-10.2	-23.6	-36	-36	-36	-36	-36	-33.6	-20.5	-9.7	-3.3	0.4	2.4	3.6	4.2	4.5	4.5	4.3	4.2	3.9
5	5.6	5.9	6.3	6.6	6.6	5.9	4.5	1.9	-2.4	-9.3	-20.9	-34.5	-36	-36	-36	-35.9	-28.9	-16.8	-7.9	-2.9	-0.5	1.8	3.1	3.5	3.9	4	4.1	3.8	3.8
4.9	5.3	5.5	5.7	5.9	5.7	4.9	3.3	0.9	-3.1	-9.1	-19	-31.3	-36	-36	-35.2	-27	-19.1	-9.7	-5.3	-2.1	-0.5	1.2	2.4	3	3.4	3.5	3.7	3.5	3.3
4.5	4.9	5	5.2	5.3	4.9	4.1	2.4	-0.5	-4.2	-10	-19.2	-29.9	-35.2	-35.7	-30.6	-20	-12.4	-7.2	-4.3	-2.4	-0.8	0.4	1.6	2.4	2.9	3.1	3.3	3.3	3.2
4.3	4.5	4.6	4.6	4.6	4.2	3.1	1.1	-1.6	-6.3	-12.5	-22	-32.6	-36	-36	-33.9	-24.6	-14.5	-10	-6.2	-4.3	-2.1	-0.8	0.7	1.5	2.2	2.6	2.9	3.1	2.9
3.9	4.2	4.3	4.2	3.9	3.3	1.8	-0.5	-3.5	-8.9	-16.1	-27.2	-36	-36	-36	-36	-36	-29.5	-18.7	-11.3	-8.2	-4.5	-2.2	-0.5	0.7	1.6	2.1	2.6	2.6	2.8
3.8	3.9	3.9	3.7	3.3	2.6	0.9	-1.5	-4.9	-10.8	-18.5	-30.4	-36	-36	-36	-36	-36	-29.5	-18.2	-11.8	-7.2	-4	-1.5	-0.4	1.1	1.6	2.2	2.4	2.6	2.6
3.5	3.5	3.5	3.2	2.8	1.9	0.4	-2.4	-6	-12.3	-20.7	-32.8	-36	-36	-36	-36	-36	-34.9	-22.8	-14.5	-9	-4.9	-2.1	-0.7	0.7	1.4	1.9	2.2	2.4	2.4
3.2	3.3	3.2	2.9	2.4	1.5	-0.5	-2.9	-6.2	-12.2	-20.2	-32	-36	-36	-36	-36	-36	-35.6	-25.3	-15.9	-10	-5.6	-2.6	-0.9	0.4	1.1	1.6	2.1	2.2	2.4
3.1	3.1	2.9	2.6	2.1	1.2	-0.5	-2.6	-5.7	-10.8	-17.5	-28.2	-36	-36	-36	-36	-36	-33	-24.1	-15.2	-9.8	-5.6	-2.8	-0.9	0.4	0.9	1.6	1.9	2.1	2.2
2.8	2.8	2.8	2.4	1.9	1.1	-0.5	-2.4	-4.6	-8.9	-14.1	-22.2	-30.2	-36	-36	-36	-36	-35.6	-27.2	-20.2	-12.8	-8.7	-4.9	-2.5	-0.9	0.4	0.9	1.5	1.8	1.9
2.6	2.6	2.6	2.3	1.9	1.1	-0.4	-1.6	-3.3	-6.6	-10.3	-15.2	-21.2	-28.8	-29.2	-29.8	-29.4	-27	-19.6	-15.1	-10	-7	-4.1	-2.1	-0.7	0.4	0.9	1.4	1.8	1.9
2.6	2.5	2.5	2.2	1.8	1.2	0.4	-0.9	-2.2	-4.3	-7.3	-11	-13.9	-17.5	-18.8	-19.3	-19.9	-17.6	-13.8	-11	-7.3	-5	-2.9	-1.4	-0.5	0.5	1	1.5	1.7	1.9

Training Set 1: 105mm Artillery

2.6 2.8 3.1 3.3 3.6 4.1 4.3 4.9 5.2 5.6 5.9 6.2 6.7 7.2 7.2 7.3 7.4 7.4 7.2 6.9 6.6 6.2 5.7 5.5 4.9 4.5 4.1 3.8 3.3 3.2 2.9
 2.8 3.1 3.3 3.8 3.9 4.5 4.8 5.3 5.7 6.3 6.7 7.3 7.9 8.3 8.4 8.6 8.6 8.6 8.3 8.2 7.7 7.3 6.7 6.2 5.5 4.9 4.5 4.2 3.8 3.3 3.1
 2.9 3.2 3.5 4.1 4.3 4.9 5.2 5.9 6.5 7.2 7.7 8.4 9.1 9.6 9.8 10.7 10.6 10.3 9.8 9.8 9 8.7 7.7 7.2 6.2 5.6 5 4.5 4 3.6 3.3
 3.2 3.3 3.8 4.3 4.5 5.3 5.6 6.6 7.2 7.9 8.6 9.7 10.3 11.1 11.7 12.4 11.8 12.3 11.4 11.3 10.6 9.8 8.4 8.2 6.9 6.2 5.5 4.9 4.3 3.9 3.5
 3.3 3.6 4.1 4.6 4.9 5.7 6.2 7.2 7.7 8.9 9.4 11 11.5 12.8 13.2 13.9 13.9 14.2 13.5 13.5 11.8 11 9.6 9 7.6 6.7 5.9 5.2 4.6 4.2 3.8
 3.3 3.8 4.2 4.9 5 6 6.5 7.6 8.3 9.6 10.1 11.7 12.7 14.5 15.1 16.2 15.9 16.5 15.5 15.2 13.7 12.4 10.7 9.8 8.4 7.3 6.3 5.6 4.9 4.3 4.1
 3.4 3.9 4.3 4.9 5.3 6.2 6.7 7.9 8.7 10 10.7 12.4 13.7 16.1 17.1 18.5 18.6 19 18.2 17.3 15.8 13.9 12 11 9.3 7.9 6.7 6 5.2 4.6 4.3
 3.5 3.9 4.3 5 5.3 6.3 6.7 7.9 8.9 10 11 13.1 14.5 17.2 18.6 20.5 20.7 21.7 20.5 19.3 17.5 15.2 13.1 11.7 10 8.4 7.2 6.3 5.5 4.8 4.5
 3.5 3.9 4.3 5 5.3 6.2 6.7 7.7 8.6 9.6 10.7 12.5 14.4 17.3 19.5 21.9 22.7 23.6 22.6 21.3 19 16.1 13.9 12.1 10.3 8.7 7.3 6.5 5.6 4.9 4.5
 3.4 3.8 4.2 4.8 5.2 5.7 6.3 7.2 7.9 8.6 9.7 11.1 13 16.5 19 21.3 23.6 24.1 23.6 22 19.7 16.6 14.2 12.3 10.6 8.9 7.4 6.6 5.6 4.9 4.6
 3.3 3.6 4.1 4.5 4.8 5.3 5.6 6.2 6.7 7.7 8.4 10.1 13.4 16.6 18.3 22.7 22.9 23.3 21.4 19.6 16.5 14.1 12.3 10.6 8.7 7.4 6.6 5.6 4.9 4.6 4.6
 3.2 3.3 3.8 4.1 4.3 4.5 4.6 4.6 5 4.5 4.9 4.2 5.6 8.2 11.7 12.1 18.9 19.9 20.3 19.6 18.1 15.4 13.2 11.4 10.1 8.2 7.2 6.5 5.5 4.9 4.6
 3.1 3.2 3.3 3.6 3.8 3.6 3.6 2.8 3.1 1.2 1.1 -0.7 -0.4 -0.5 3.5 3.3 11.1 13.9 15.4 14.4 14.7 13.2 11.7 10.1 9.4 7.7 6.7 6 5.2 4.6 4.5
 2.8 2.9 2.9 2.8 2.9 2.2 1.9 0.4 -0.5 -3.1 -4.3 -7.2 -7.6 -10.6 -7.9 -9.4 -1.1 3.2 6 7 7.9 9.6 9 8.2 7.6 6.7 6 5.5 4.9 4.3 4.2
 2.4 2.5 2.5 2 2.1 0.9 0.8 -2.1 -3.1 -6.2 -9.6 -14.5 -15.5 -20.7 -22 -20.9 -13.8 -11 -5.3 -1.5 1.9 5.2 6.2 5.6 6.6 5.6 5.3 4.9 4.5 3.7 3.9
 2.1 2.1 1.9 1.2 1.1 -0.4 -1 -3.5 -6 -12.1 -13.5 -20.5 -25.3 -29.9 -32.8 -33.7 -29.5 -21.6 -16.1 -9.4 -3.3 0.5 2.5 3.3 3.8 4.3 4.2 4.2 3.9 3.5 3.6
 1.8 1.6 1.4 0.5 0.4 -1.2 -2.5 -5.7 -9 -15.5 -18 -26.5 -32.6 -36 -36 -36 -36 -32.2 -26.5 -16.9 -10.3 -3.8 -1.1 0.8 1.9 2.9 3.3 3.3 3.3 3.2 3.2
 1.5 1.4 0.9 -0.5 -0.7 -2.5 -3.8 -7.9 -11.4 -18.8 -23 -32.5 -36 -36 -36 -36 -36 -36 -32.8 -25 -16.8 -8.2 -4.9 -1.4 0.4 1.6 2.4 2.5 3.8 2.8 2.8
 1.2 1.1 0.4 -0.8 -1.6 -3.3 -5.5 -8.9 -14.8 -20.9 -27.4 -34.5 -36 -36 -36 -36 -36 -36 -36 -31.1 -22.2 -12.5 -8.2 -4.1 -1.2 0.4 1.6 1.9 3.2 2.4 2.4
 1.1 0.7 0.4 -1.1 -2.1 -4.2 -5.9 -10 -15.8 -23.1 -28.8 -35.6 -36 -36 -36 -36 -36 -36 -36 -33.6 -26.1 -14.7 -10.6 -5.6 -2.6 -0.7 0.8 1.1 1.6 1.9 2.1
 0.9 0.4 -0.5 -1.5 -2.6 -4.8 -6.6 -10.8 -16.2 -24.1 -29.2 -35.9 -36 -36 -36 -36 -36 -36 -36 -34 -27.9 -16.6 -11.5 -6.9 -3.5 -1.5 -0.4 0.5 1.1 1.6 1.8
 0.8 0.4 -0.5 -1.6 -2.6 -4.8 -6.7 -10.6 -14.9 -23.4 -28.2 -34.9 -36 -36 -36 -36 -36 -36 -36 -33 -27.7 -16.8 -11.8 -7.4 -4.5 -2.1 -0.7 0.4 0.7 1.2 1.5
 0.8 0.4 -0.5 -1.6 -2.6 -4.6 -6.3 -10 -14.8 -21.4 -25.8 -32.5 -36 -36 -36 -36 -36 -36 -36 -31.1 -25.6 -15.8 -11.7 -7.3 -4.8 -2.4 -0.8 -0.5 0.4 1.1 1.2
 0.7 0.4 -0.5 -1.6 -2.5 -4.3 -5.9 -8.9 -13 -18.8 -21.9 -27.7 -33 -36 -36 -36 -36 -36 -36 -34 -31 -22.6 -14.5 -11 -6.9 -4.6 -2.4 -0.9 -0.5 0.4 0.9 1.1
 0.7 0.4 -0.5 -1.4 -2.2 -3.8 -5.3 -7.9 -11.4 -15.6 -18.6 -23.1 -28.2 -31.8 -34.9 -33.6 -34.3 -32 -29.4 -27.8 -19.3 -12.7 -9.6 -6.2 -4.2 -2.2 -0.9 -0.5 0.4 0.8 0.9
 0.7 0.4 -0.5 -1.1 -1.9 -3.2 -4.5 -6.5 -9.4 -13 -15.1 -18.5 -23.2 -25 -27.7 -26.7 -28.1 -25.7 -21.9 -23 -15.2 -10.4 -8.2 -5.3 -3.8 -1.9 -0.8 -0.5 0.4 0.8 0.9
 0.7 0.4 -0.5 -0.8 -1.5 -2.5 -3.8 -5 -7.7 -9.7 -12.1 -14.4 -17.8 -18.2 -21.3 -20.9 -20.6 -20.3 -17.3 -19 -11.7 -8.6 -6.6 -4.3 -3.1 -1.6 -0.7 -0.4 0.4 0.8 0.9
 0.8 0.5 0.4 -0.5 -1.1 -1.9 -2.9 -3.9 -5.7 -7 -8.9 -10.6 -13 -13.7 -14.8 -15.4 -14.7 -13.8 -13.5 -14.5 -9.3 -6.2 -5 -3.4 -2.6 -1.2 -0.7 0.4 0.4 0.8 0.9
 0.8 0.7 0.4 -0.5 -0.7 -1.4 -2.1 -2.9 -4.5 -5.7 -6.7 -8.2 -9.4 -10 -11.4 -10.7 -11.1 -10.3 -10 -10.1 -6.7 -4.5 -3.9 -2.6 -1.6 -0.9 -0.5 0.4 0.4 0.8 0.9
 0.9 0.7 0.4 0.4 -0.5 -0.9 -1.6 -2.2 -3.3 -4.2 -4.9 -5.9 -7.2 -7.3 -8.2 -7.9 -8 -7.4 -7.4 -8.3 -5.2 -3.6 -2.8 -1.9 -1.2 -0.7 -0.4 0.5 0.5 0.8 0.9
 1 0.8 0.5 0.4 -0.5 -0.5 -0.9 -1.4 -2.2 -3.1 -3.5 -3.9 -4.8 -5.2 -5.6 -5.5 -5.6 -5.2 -5 -6.5 -3.5 -2.4 -1.8 -1.1 -0.8 -0.5 0.4 0.5 0.6 0.8 0.9
 1.1 0.9 0.8 0.5 0.4 -0.4 -0.7 -0.8 -1.4 -1.8 -2.4 -2.6 -3.2 -3.3 -3.9 -3.6 -3.8 -3.6 -3.3 -4.2 -2.4 -1.6 -1.2 -0.7 -0.5 0.4 0.4 0.8 0.7 0.9 0.9
 1.2 1.1 0.9 0.8 0.4 0.4 -0.5 -0.5 -0.8 -1.1 -1.5 -1.6 -2.2 -2.1 -2.6 -2.5 -2.6 -2.5 -2.4 -2.9 -1.5 -0.9 -0.7 -0.5 -0.4 0.4 0.5 0.8 0.8 0.9 1.1
 1.2 1.1 0.9 0.9 0.5 0.5 0.4 0.4 -0.5 -0.9 -0.8 -0.9 -1.2 -1.2 -1.5 -1.5 -1.6 -1.5 -1.2 -1.9 -0.8 -0.5 -0.5 0.4 0.4 0.5 0.7 0.8 0.9 0.9 1.1
 1.2 1.2 1.1 1 0.8 0.7 0.5 0.4 0.4 -0.4 -0.5 -0.5 -0.7 -0.7 -0.8 -0.8 -0.9 -0.8 -0.7 -0.7 -0.5 -0.4 0.4 0.4 0.5 0.7 0.8 0.8 0.9 1.1 1.1

Training Set 1: 105mm HEAT

3.5	4.1	4.7	5.5	6.6	7.7	9	10.8	12.7	15.5	18.1	21	23.6	25.8	27.1	26.8	25.7	23.7	21.3	18.3	15.9	13	11.7	9.6	8	7	5.9	4.8	4.3	3.8	3.3
3.6	4.2	4.9	5.6	6.9	8	9.6	11.7	13.9	16.9	20	23.6	26.8	29.2	30.8	31.1	29.8	27.5	24.6	20.9	17.8	14.7	12.8	10.4	8.7	7.3	6.2	5.3	4.5	3.9	3.5
3.8	4.3	5	5.9	7.1	8.4	10	12.3	14.7	17.8	21.4	25.4	28.9	31.8	33	33.6	32	30.4	26.8	23	19.3	15.9	13.7	11.1	9.1	7.9	6.6	5.6	4.8	4.2	3.6
3.8	4.3	5	5.8	7	8.4	10	12.3	14.5	17.8	21.3	25.3	29.2	31.8	33.6	33.9	33	30.9	27.9	23.7	20	16.6	14.2	11.5	9.4	7.9	6.7	5.7	4.9	4.3	3.8
3.8	4.3	4.9	5.7	6.9	8.2	9.6	11.7	13.8	16.9	19.7	23.4	26.8	29.2	31.3	31.9	32.2	29.4	27.1	23	19.7	16.5	14.1	11.5	9.6	8	6.7	5.7	4.9	4.3	3.8
3.6	4.2	4.8	5.5	6.6	7.6	8.9	10.6	12.4	14.4	16.8	18.9	21.6	22.9	25.5	25.8	27.4	25.3	24.4	21.3	18.6	15.8	13.7	11.3	9.4	7.9	6.7	5.7	4.9	4.3	3.8
3.5	4.1	4.5	5	5.9	6.7	7.9	8.9	10.1	11	12.3	11.8	14.1	12.4	15.1	15.1	17.8	17.8	19.2	17.5	16.6	14.4	12.5	10.6	9	7.6	6.6	5.6	4.8	4.3	3.8
3.3	3.8	4.2	4.6	5.2	5.6	6.3	6.6	6.9	6.6	6	3.5	3.3	-0.8	2.1	0.7	6.5	8.6	13.4	12.4	13.4	11.8	11.3	9.4	8.3	7.2	6.2	5.5	4.8	4.2	3.8
3.2	3.5	3.8	4.1	4.3	4.5	4.6	4	3.2	1.2	-0.8	-6.2	-8.4	-15.2	-13.2	-15.2	-5.6	-1.8	4.9	6.7	9.4	8.9	9.1	8.2	7.4	6.6	5.7	5.1	4.5	4.1	3.6
3.1	3.2	3.3	3.3	3.5	3.3	2.9	1.5	-0.4	-3.6	-7.9	-15.2	-21.3	-28.8	-28.7	-27.5	-20.2	-12.4	-3.9	0.8	4.6	5.2	7	6.6	6.5	5.7	5.3	4.7	4.3	3.9	3.5
2.8	2.9	2.9	2.8	2.6	2.1	1.1	-0.9	-3.1	-8.7	-13.8	-24.3	-31.6	-36	-36	-36	-31.1	-25	-12.5	-5	0.5	2.5	4.6	5	5.3	5	4.9	4.5	4.1	3.8	3.3
2.6	2.6	2.5	2.4	1.8	0.9	-0.5	-2.9	-6	-12.4	-19.6	-30.9	-36	-36	-36	-36	-36	-32.9	-21	-11	-3.9	-0.5	2.2	3.3	4.2	4.3	4.3	4.1	3.8	3.5	3.2
2.4	2.4	2.2	1.9	1.2	0.4	-1.4	-4.2	-8.3	-14.9	-23.4	-33.9	-36	-36	-36	-36	-36	-27.4	-16.1	-7.6	-3.1	0.4	2.1	3.3	3.6	3.8	3.6	3.5	3.3	3.2	
2.2	2.2	2	1.5	0.8	-0.5	-2.3	-5.3	-9.8	-16.8	-25.7	-35.6	-36	-36	-36	-36	-36	-30.5	-19.3	-10.1	-5	-1.2	0.8	2.4	2.9	3.3	3.3	3.2	3.1	2.9	
2.1	2.1	1.7	1.2	0.5	-0.7	-2.6	-5.7	-10	-16.9	-25.5	-35.5	-36	-36	-36	-36	-36	-32	-20.7	-11.4	-6.3	-2.2	0.4	1.6	2.4	2.8	2.9	3.1	2.9	2.8	
2.1	1.9	1.6	1.2	0.4	-0.7	-2.5	-5.2	-9	-15.2	-22.9	-32.6	-36	-36	-36	-36	-36	-30.8	-19.6	-11.4	-6.3	-2.6	-0.5	1.2	2.1	2.5	2.8	2.8	2.8	2.6	
2	1.8	1.5	1.2	0.4	-0.6	-2.2	-4.6	-7.8	-12.4	-19.6	-27.5	-35.2	-36	-36	-36	-36	-33.9	-27.2	-16.5	-9.6	-5.5	-2.4	-0.5	1.1	1.9	2.4	2.6	2.6	2.6	2.6
1.9	1.8	1.5	1.2	0.5	-0.5	-1.6	-3.5	-6	-9.3	-14.6	-20.6	-27.7	-31.9	-33.9	-35	-29.3	-24	-19.9	-11.8	-7.2	-3.9	-1.6	0.4	1.2	1.9	2.3	2.4	2.5	2.5	2.5
1.9	1.8	1.6	1.3	0.8	0.4	-0.8	-2.1	-3.8	-5.6	-9	-12.7	-19.3	-18.8	-22	-21.3	-19	-16.4	-13.5	-7.9	-4.9	-2.5	-0.8	0.5	1.3	1.9	2.2	2.4	2.4	2.4	2.4
1.8	1.7	1.6	1.4	0.9	0.5	-0.5	-1.1	-2.2	-3.7	-5.6	-7.7	-10.7	-10.7	-12.7	-11.7	-10.7	-7.9	-6.9	-4.1	-2.6	-0.8	0.4	0.9	1.5	1.9	2.2	2.2	2.4	2.4	2.2
1.8	1.6	1.6	1.4	1.1	0.7	0.4	-0.7	-1.2	-2.1	-3.3	-4.3	-5.9	-6	-6.6	-5.9	-5.3	-4.1	-3.2	-1.2	-0.7	0.4	0.8	1.3	1.6	1.9	2.1	2.2	2.2	2.2	2.2
1.7	1.6	1.5	1.4	1.1	0.8	0.4	-0.5	-0.9	-1.6	-2.6	-3.8	-4.6	-5.9	-5.9	-6.2	-4.5	-3.2	-1.8	-0.9	-0.4	0.4	0.9	1.4	1.6	1.9	2.1	2.1	2.1	2.1	2.1
1.6	1.6	1.5	1.2	0.9	0.7	0.4	-0.5	-1.1	-2.2	-3.3	-5.3	-6.6	-9	-9.4	-10.1	-7.3	-5.5	-3.2	-1.9	-0.5	0.4	0.7	1.1	1.5	1.6	1.9	1.9	2.1	2.1	2.1
1.6	1.5	1.4	1.1	0.8	0.4	-0.5	-0.9	-1.9	-3.6	-5.5	-9	-10.3	-14.5	-16	-17.5	-13.2	-10.1	-6.9	-4.5	-2.1	-1	-0.4	0.5	1.1	1.4	1.6	1.8	1.9	1.9	1.9
1.5	1.4	1.2	0.9	0.5	0.4	-0.7	-1.6	-3.1	-5.1	-7.9	-12.5	-15.6	-20.9	-23.4	-25	-19	-16.1	-11.1	-7.2	-3.8	-2.5	-0.8	-0.4	0.7	1.1	1.5	1.6	1.6	1.8	1.8
1.5	1.3	1.1	0.8	0.4	-0.5	-1.1	-2.4	-4.2	-6.7	-10.3	-15.8	-20.6	-26.5	-30.2	-30.6	-26	-21.3	-15.5	-10.7	-6.2	-3.9	-1.8	-0.7	0.4	0.7	1.2	1.4	1.5	1.6	1.6
1.4	1.2	0.9	0.7	0.4	-0.7	-1.6	-3.2	-5.3	-8.3	-12.5	-18.9	-24.6	-31.2	-34.2	-34.7	-31.6	-26.4	-19.5	-13.5	-8.3	-5.6	-3	-1.4	-0.5	0.4	0.9	1.1	1.4	1.5	1.6
1.2	1.1	0.8	0.4	-0.5	-0.9	-2.2	-3.8	-6.2	-9.4	-13.9	-20.7	-27.5	-33.2	-36	-36	-34.3	-29.6	-22	-15.6	-9.8	-6.7	-3.8	-1.9	-0.8	-0.4	0.7	0.9	1.2	1.4	1.5
1.2	0.9	0.8	0.4	-0.5	-1.1	-2.4	-3.9	-6.6	-9.7	-14.4	-21.2	-27.9	-33.2	-36	-36	-35	-30.8	-23.7	-16.8	-11	-7.6	-4.5	-2.5	-1.1	-0.5	0.4	0.8	1.1	1.2	1.5
1.1	0.9	0.7	0.4	-0.5	-1.2	-2.6	-4.1	-6.6	-9.7	-14.1	-20.5	-26.8	-31.8	-35	-35.2	-33.6	-29.2	-22.4	-16.1	-11	-7.6	-4.6	-2.6	-1.3	-0.5	0.4	0.7	0.9	1.2	1.4
1.1	0.9	0.5	0.4	-0.5	-1.2	-2.6	-3.9	-6.2	-9	-13.1	-18.2	-24.3	-28.4	-31.8	-31.9	-30.8	-26	-20.5	-14.8	-10.1	-7	-4.5	-2.6	-1.4	-0.5	0.4	0.6	0.9	1.1	1.4
1.1	0.9	0.5	0.4	-0.5	-1.1	-2.2	-3.5	-5.5	-7.9	-11.1	-15.2	-20.3	-23.1	-26.4	-26.8	-25.8	-21.7	-17.8	-13	-9	-6.6	-4.2	-2.4	-1.3	-0.5	0.4	0.5	0.9	1.1	1.2
1.1	0.9	0.6	0.4	-0.5	-0.9	-1.9	-3	-4.7	-6.6	-9.3	-12.1	-15.9	-17.5	-21.2	-20.2	-19.9	-17.5	-14.2	-10.4	-7.7	-5.5	-3.6	-2.1	-1.1	-0.5	0.4	0.5	0.9	1.1	1.2
1.1	0.9	0.7	0.4	-0.5	-0.7	-1.5	-2.4	-3.9	-5	-7.2	-9.4	-12.4	-13.2	-15.5	-14.7	-14.4	-12.7	-10.7	-8	-6	-4.3	-2.8	-1.6	-0.8	-0.5	0.4	0.5	0.9	1.1	1.2
1.1	0.9	0.7	0.4	-0.5	-1.1	-1.8	-2.8	-3.9	-5.3	-7	-8.4	-9.4	-10.3	-10.4	-10	-9	-7.7	-6	-4.8	-3.3	-2.1	-1.2	-0.7	-0.5	0.4	0.7	0.9	1.1	1.2	

Training Set 2: 3-5in Rocket

1.5	1.6	1.8	2	2.2	2.4	2.6	2.8	3.1	3.3	3.5	3.8	3.9	3.9	3.9	3.8	3.6	3.5	3.3	3	2.4	2.6	2.4	2.4	1.9	1.9	1.4	1.4	1.1	1.2	1.2		
1.6	1.8	1.9	2.1	2.4	2.6	2.9	3.3	3.6	3.9	4.2	4.5	4.6	4.6	4.6	4.3	4.2	3.9	3.6	2.8	2.9	2.6	2.4	2.2	2	1.5	1.5	1.2	1.4	1.2			
1.8	1.9	2.1	2.4	2.6	3.1	3.3	3.8	4.2	4.5	4.5	5.5	5.6	5.7	5.7	5.6	5.3	5	4.5	4.2	3.5	3.3	3.1	2.6	2.4	2.1	1.6	1.6	1.5	1.5	1.4		
1.9	2.1	2.2	2.6	2.9	3.3	3.8	4.3	4.9	5.5	5.9	6.5	6.7	7	6.9	6.7	6.5	6	5.3	4.9	4.1	3.9	3.3	2.9	2.6	2.4	1.9	1.8	1.6	1.5	1.4		
1.9	2.2	2.5	2.8	3.3	3.7	4.2	4.9	5.6	6.3	7.2	7.7	8.3	8.4	8.4	8.2	7.6	7.3	6.5	5.7	4.9	4.3	3.8	3.3	2.9	2.6	2.2	1.9	1.8	1.5	1.5		
2.1	2.2	2.6	3.1	3.5	4.1	4.6	5.5	6.5	7.4	8.4	9.6	10	10.7	10.1	10.1	9.3	8.9	7.4	6.7	5.6	4.9	4.2	3.6	3.2	2.6	2.4	2.1	1.8	1.5	1.5		
2.1	2.4	2.7	3.2	3.8	4.3	5.2	6.2	7.3	8.4	9.6	11.1	11.8	12.7	12.4	12.3	11	10.1	8.6	7.7	6.3	5.6	4.6	3.9	3.3	2.9	2.4	2.1	1.9	1.6	1.6		
2.1	2.4	2.8	3.3	3.9	4.6	5.5	6.7	8	9.7	11.3	12.8	14.1	14.8	14.5	14.5	13	11.8	9.8	8.6	7.2	6	5	4.2	3.5	3.1	2.5	2.2	1.9	1.6	1.6		
2.1	2.4	2.8	3.3	3.9	4.8	5.7	7	8.4	10.3	12.4	14.3	15.8	17	16.6	16.6	14.9	13.2	11	9.4	7.7	6.6	5.3	4.3	3.6	3.2	2.6	2.2	2.1	1.8	1.6		
2.1	2.4	2.8	3.3	3.9	4.6	5.6	7	8.7	10.6	12.7	14.9	16.9	18.2	18.5	17.9	16.2	14.1	11.8	9.8	8	6.7	5.5	4.5	3.8	3.2	2.6	2.4	2.1	1.8	1.6		
2.1	2.2	2.6	3.1	3.8	4.3	5.3	6.6	8.2	9.8	11.7	13.9	16.2	17.5	18.6	17.9	16.4	14.1	12.1	9.8	8.2	6.7	5.5	4.5	3.8	3.2	2.6	2.2	2.1	1.8	1.6		
1.9	2.1	2.4	2.8	3.3	3.9	4.6	5.6	6.7	8.2	9.4	11.3	13.2	14.4	16.5	15.5	14.8	13	11.1	9.6	7.6	6.5	5.2	4.3	3.6	3.2	2.6	2.2	2.1	1.8	1.6		
1.8	1.9	2.2	2.4	2.8	3.1	3.8	4.1	5	5.5	6.2	7	7.9	11.3	10.7	11.4	9.8	9.8	8	6.9	5.7	4.8	3.9	3.3	3	2.5	2.2	1.9	1.8	1.6			
1.6	1.6	1.8	1.8	2.2	2.1	2.4	2.2	1.8	1.2	0.4	-1.1	-1	-1.2	1.9	2.9	5	5.2	6.2	5.7	5.2	4.8	3.9	3.3	3	2.6	2.2	2.1	1.8	1.6	1.5		
1.4	1.4	1.4	1.2	1.2	0.7	0.7	-0.5	-1.6	-3.9	-6.2	-11.1	-11	-12.3	-9.7	-6.5	-3.6	-0.5	2.5	3	3.5	3.2	3.1	2.8	2.6	2.4	2.1	1.8	1.6	1.6	1.5		
1.1	1.1	0.9	0.8	0.4	0.4	-0.9	-2.2	-5	-8.3	-12	-19	-20.6	-23.6	-19.2	-16.9	-10.7	-5.5	-1.9	-0.5	1.2	1.8	2.1	2.1	2.1	1.9	1.8	1.6	1.6	1.5	1.5		
0.9	0.8	0.5	0.4	-0.5	-1.2	-2.6	-4.3	-7.5	-11.7	-16.7	-22.9	-27.8	-29.5	-26.8	-24.8	-16.5	-10.6	-5.7	-3.1	-0.7	0.4	1.1	1.4	1.5	1.5	1.6	1.5	1.5	1.4	1.4		
0.8	0.5	0.4	-0.5	-0.9	-2.1	-3.5	-6	-9.4	-13.9	-19.9	-26.4	-31.6	-33.3	-31.3	-28.7	-20.7	-14.4	-8.6	-5.5	-2.4	-1	0.4	0.7	1.1	1.1	1.4	1.3	1.4	1.2	1.2		
0.7	0.4	0.4	-0.5	-1.3	-2.4	-4.1	-6.2	-10	-14.8	-20.1	-26.5	-32.2	-33.9	-33.3	-30.5	-23.7	-17.2	-11.3	-7.6	-3.9	-2.4	-0.7	0.4	0.5	0.8	1.1	1.1	1.2	1.1	1.2		
0.5	0.4	-0.5	-0.7	-1.5	-2.6	-4.5	-6.7	-10.4	-14.5	-20.2	-25.4	-31.6	-32.9	-32.6	-29.6	-23.7	-17.9	-12.1	-8.3	-4.8	-3.1	-1.2	-0.5	0.4	0.5	0.8	0.9	1.1	1.1	1.1		
0.4	0.4	-0.5	-0.8	-1.6	-2.8	-4.5	-6.6	-10	-13.4	-18.6	-23	-28.8	-29.5	-30.4	-26.8	-22.2	-16.9	-12.3	-8.4	-5.2	-3.3	-1.6	-0.7	-0.5	0.4	0.7	0.8	0.9	0.9	1.1		
0.4	0.4	-0.5	-0.8	-1.6	-2.6	-4.2	-6	-9	-11.7	-16.2	-19	-24.6	-24.6	-25.5	-22.4	-19	-15.2	-11.3	-8.2	-5.2	-3.3	-1.8	-0.9	-0.5	0.4	0.5	0.7	0.9	0.9	1		
0.4	0.4	-0.5	-0.8	-1.5	-2.4	-3.8	-5.3	-7.7	-9.8	-13.2	-15.8	-19	-19.6	-20.6	-18.1	-16.2	-12.5	-9.7	-7.2	-4.8	-3.2	-1.8	-0.9	-0.5	0.4	0.4	0.7	0.8	0.9	0.9		
0.4	0.4	-0.5	-0.7	-1.2	-2.1	-3.1	-4.3	-6.2	-7.9	-10.7	-11.8	-13.2	-14.8	-16.6	-14.1	-13	-9.7	-8	-6.2	-4.2	-2.9	-1.6	-0.9	-0.5	0.4	0.4	0.7	0.8	0.8	0.9		
0.4	0.4	-0.5	-0.7	-1.1	-1.6	-2.6	-3.5	-5	-6.1	-8.3	-9.1	-11.4	-11.1	-12.3	-10.6	-10	-7.6	-6.2	-4.8	-3.3	-2.4	-1.4	-0.7	-0.5	0.4	0.4	0.7	0.8	0.8	0.9		
0.4	0.4	-0.4	-0.5	-0.8	-1.2	-2.1	-2.8	-4	-4.8	-6.2	-6.9	-8.3	-8.3	-8.9	-7.6	-7.3	-5.7	-4.9	-3.8	-2.6	-1.8	-0.9	-0.7	-0.5	0.4	0.4	0.7	0.8	0.8	0.9		
0.5	0.4	0.4	-0.5	-0.7	-0.9	-1.5	-2.1	-2.9	-3.5	-4.5	-4.8	-5.6	-5.7	-6.3	-5.3	-5.3	-4.1	-3.8	-2.9	-1.9	-1.4	-0.8	-0.5	-0.4	0.4	0.5	0.7	0.7	0.8	0.9		
0.5	0.4	0.4	-0.4	-0.5	-0.7	-0.9	-1.4	-2.1	-2.5	-3.2	-3.3	-4.1	-3.9	-4.6	-3.6	-3.3	-2.7	-2.5	-1.9	-1.5	-0.9	-0.7	-0.5	0.4	0.4	0.5	0.7	0.8	0.8	0.9		
0.6	0.5	0.4	0.4	-0.5	-0.5	-0.7	-0.9	-1.3	-1.6	-2.1	-2.2	-2.6	-2.6	-3.1	-2.4	-2.4	-1.9	-1.8	-1.2	-0.8	-0.7	-0.5	-0.4	0.4	0.4	0.5	0.7	0.7	0.8	0.9		
0.7	0.5	0.4	0.4	0.4	-0.5	-0.5	-0.7	-0.8	-1.1	-1.4	-1.5	-1.8	-1.8	-2.1	-1.6	-1.5	-1.2	-1	-0.8	-0.7	-0.5	-0.4	0.4	0.4	0.4	0.5	0.8	0.7	0.8	0.9		
0.8	0.6	0.5	0.4	0.4	0.4	-0.5	-0.5	-0.5	-0.7	-0.8	-0.8	-0.9	-0.9	-1.2	-0.9	-0.9	-0.7	-0.7	-0.5	-0.5	-0.5	-0.4	0.4	0.4	0.4	0.5	0.7	0.8	0.7	0.9	0.9	
0.8	0.7	0.5	0.5	0.4	0.4	0.4	0.4	-0.5	-0.5	-0.7	-0.5	-0.6	-0.7	-0.7	-0.7	-0.7	-0.5	-0.5	-0.5	-0.5	-0.4	0.4	0.4	0.4	0.4	0.4	0.7	0.7	0.8	0.7	0.9	0.9
0.8	0.8	0.7	0.5	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.7	0.7	0.8	0.7	0.9	0.8	
0.8	0.8	0.5	0.7	0.5	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.5	0.4	0.4	0.8	0.8	0.9	0.7	0.9	0.9	
0.8	0.8	0.5	0.8	0.7	0.7	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.4	0.6	0.5	0.4	0.4	0.8	0.8	0.9	0.8	0.9	0.9	0.9		

Training Set 2: 60mm Mortar

5.1	3.3	3.8	4.2	4.8	5.5	6	6.7	7.4	8.3	9	9.3	9.8	10	9.6	9.4	8.7	8.2	7.7	6.7	6.3	5.7	5.2	4.6	4.2	3.8	3.3	3.1	2.6	2.4	2.2
3.3	3.6	4.2	4.6	5.5	6.2	6.9	8.2	9	10	11	11.7	12.1	12.1	11.7	11.4	10.6	10	9	8	7.3	6.6	5.9	5.2	4.6	4.1	3.6	3.3	2.9	2.5	2.4
3.5	3.9	4.5	5	6	7.2	8.2	9.4	10.6	12.3	13.4	14.4	15.2	15.2	14.7	13.9	12.5	11.7	10.6	9.6	8.4	7.4	6.7	5.7	5.2	4.5	3.9	3.5	3.1	2.6	2.5
3.8	4.2	4.9	5.6	6.6	7.9	8.9	10.8	12	14.4	15.8	17.3	18.2	18.1	17.5	16.5	15.1	13.9	12.4	11.1	9.6	8.6	7.4	6.5	5.6	4.9	4.3	3.8	3.3	2.8	2.6
3.9	4.5	5.2	6	7.2	8.6	10	12.3	13.8	16.8	18.9	20.7	21.9	21.9	20.6	19.6	17.8	16.1	14.4	12.8	10.8	9.8	8.3	7.2	6.2	5.2	4.6	3.9	3.5	3.1	2.6
4.1	4.6	5.5	6.2	7.7	9.1	10.7	13.4	15.2	18.8	21.9	24.3	25.5	25.5	24.6	22.7	20.3	18.3	16.5	14.5	12.3	10.8	9.1	7.9	6.7	5.6	4.9	4.2	3.6	3.3	2.8
4.2	4.8	5.6	6.6	8	9.7	11.7	14.5	17	20.5	24	26.8	28.7	29.9	27.9	26.3	23.1	21	18.3	16.4	13.5	11.8	10	8.4	7.2	5.9	5	4.3	3.8	3.3	2.9
4.2	4.8	5.6	6.6	8.2	9.8	11.8	14.7	17.5	21.2	25	27.9	30.4	30.9	30.1	28.7	25.7	23.3	20.5	17.8	14.7	12.7	10.6	8.9	7.4	6.2	5.3	4.5	3.9	3.3	3.1
4.2	4.8	5.6	6.5	8	9.6	11.7	14.4	16.9	20.3	24.1	27	29.2	30.1	30.1	28.5	27.1	24.4	21.4	18.5	15.4	13	10.8	9	7.6	6.3	5.3	4.5	3.9	3.5	3.1
4.1	4.6	5.5	6.2	7.6	8.9	11	13.1	15.6	18.2	20.7	22.6	24.8	24.3	26	23.8	25	23.1	20.9	17.9	15.2	12.7	10.8	8.7	7.6	6.2	5.3	4.5	3.9	3.5	3.1
3.9	4.3	5	5.6	6.7	7.9	9.6	11.1	12.5	13.9	15.5	14.5	14.7	13.4	17.8	15.2	20.2	18.6	18.5	16.2	14.5	12.1	10.4	8.4	7.3	6	5.2	4.3	3.9	3.3	3.1
3.6	3.9	4.5	5	5.9	6.7	7.7	8.4	9.1	9.1	6.5	6.6	3.5	1.2	3.6	3.3	11.3	10.7	13.9	13.4	12.5	10.6	9.4	7.9	6.9	5.6	4.9	4.2	3.8	3.3	2.9
3.3	3.6	3.9	4.3	5	5.2	5.6	5.5	5	2.8	-0.9	-4.3	-8.4	-14.5	-11.3	-9.6	-1.6	1.9	8.2	9	9.8	8.4	8	6.7	6.2	5.2	4.6	3.9	3.6	3.2	2.8
3	3.2	3.3	3.5	3.8	3.3	3.3	0.9	0.4	-3.3	-9.3	-15.6	-23.3	-30.1	-30.9	-25.8	-12.4	-7	1.8	3.8	5.6	5.7	6	5.2	5	4.3	4.1	3.6	3.3	3	2.6
2.6	2.6	2.6	2.5	2.4	1.6	1.1	-1.5	-3.9	-9.8	-17.5	-25.8	-32.5	-36	-36	-36	-24.8	-18.3	-6.7	-2.6	1.2	2.9	3.8	3.8	4.1	3.6	3.5	3.2	3.1	2.9	2.5
2.2	2.1	1.9	1.6	1.1	0.4	-0.9	-4.9	-8	-15.2	-23.4	-32.5	-36	-36	-36	-36	-26.4	-14.2	-8.4	-2.4	0.4	1.6	2.2	2.8	2.8	3.1	2.8	2.6	2.5	2.4	
1.8	1.6	1.5	0.8	-0.5	-1.2	-3.2	-6.7	-12	-19.3	-28.7	-35.9	-36	-36	-36	-36	-36	-32.9	-21.3	-13.4	-6.3	-2.6	-0.7	0.8	1.6	2.1	2.4	2.4	2.4	2.2	2.2
1.5	1.2	0.9	0.4	-0.9	-2.4	-5	-9.3	-14.8	-23.6	-32.3	-36	-36	-36	-36	-36	-36	-35.6	-25.7	-17.5	-9.1	-5	-2.1	-0.5	0.8	1.4	1.9	1.9	2.1	2.1	2.1
1.2	0.9	0.5	-0.5	-1.4	-3.2	-6	-10	-15.8	-24	-33.2	-36	-36	-36	-36	-36	-36	-29.8	-20.9	-12	-7.2	-3.9	-1.6	-0.5	0.6	1.4	1.5	1.6	1.8	1.9	
1.1	0.7	0.4	-0.7	-2	-3.8	-6.6	-10.8	-16.3	-24.7	-33.5	-36	-36	-36	-36	-36	-36	-30.6	-22.3	-13	-8.4	-4.8	-2.5	-0.7	0.4	0.9	1.1	1.5	1.6	1.6	
0.9	0.5	-0.5	-0.9	-2.2	-4.1	-6.6	-10.8	-16.1	-23.6	-32.3	-36	-36	-36	-36	-36	-36	-30	-22	-13.8	-8.9	-5.5	-2.9	-1.2	-0.5	0.5	0.9	1.2	1.4	1.6	
0.8	0.4	-0.5	-0.9	-2.2	-4	-6.5	-10.1	-15.1	-21.3	-19.2	-35	-36	-36	-36	-36	-36	-34.9	-28.1	-20.7	-13.2	-8.9	-5.6	-3.2	-1.5	-0.6	0.4	0.8	1.1	1.2	1.5
0.7	0.4	-0.5	-0.9	-2.1	-3.8	-5.9	-9.1	-13.4	-18.5	-25	-30.9	-36	-36	-36	-36	-36	-31.1	-24	-18.3	-12.1	-8.2	-5.5	-3.2	-1.6	-0.7	0.4	0.6	0.9	1.2	1.4
0.7	0.4	-0.5	-0.9	-1.9	-3.2	-5	-7.9	-11.3	-15.2	-20.2	-24.7	-30.9	-35	-35.4	-32.6	-31.3	-24.7	-19.6	-15.5	-10.6	-7.3	-4.9	-2.8	-1.6	-0.7	0.4	0.5	0.9	1.1	1.2
0.7	0.4	-0.5	-0.8	-1.6	-2.8	-4.3	-6.6	-9.3	-12.4	-16.2	-19.3	-24.3	-26	-27.9	-25.4	-23.8	-19.6	-15.1	-11.8	-8.4	-6	-4.2	-2.4	-1.4	-0.5	0.4	0.5	0.8	1.1	1.2
0.8	0.4	-0.5	-0.7	-1.2	-2.2	-3.5	-5.3	-7.3	-9.4	-12.1	-14.2	-18.5	-18.5	-20.1	-18.8	-18.2	-14.8	-12	-9.1	-6.7	-4.9	-3.3	-1.9	-1.1	-0.5	0.4	0.5	0.8	1.1	1.2
0.8	0.5	0.4	-0.5	-0.9	-1.6	-2.6	-4.1	-5.5	-7	-8.9	-10.7	-13	-13.2	-14.2	-13.5	-13	-11.3	-8.9	-7.2	-5.2	-3.8	-2.6	-1.5	-0.8	-0.5	0.4	0.6	0.9	1.1	1.2
0.8	0.7	0.4	-0.4	-0.7	-1.2	-1.9	-2.9	-4.2	-5	-6.3	-7.4	-9	-9.3	-9.8	-9.3	-9.1	-7.7	-6.6	-5.2	-3.9	-2.6	-1.8	-1.1	-0.5	0.4	0.4	0.7	0.9	1.1	1.2
0.9	0.8	0.4	0.4	-0.5	-0.7	-1.2	-1.9	-2.8	-3.8	-4.3	-4.9	-6.2	-6.5	-6.9	-6.6	-6.6	-5.5	-4.5	-3.8	-2.8	-1.8	-1.2	-0.7	-0.5	0.4	0.5	0.8	0.9	1.1	1.2
0.9	0.8	0.5	0.4	0.4	-0.5	-0.7	-1.2	-1.9	-2.4	-2.9	-3.5	-4.2	-4.3	-4.6	-4.2	-4.3	-3.8	-2.9	-2.5	-1.6	-1.1	-0.7	-0.5	0.4	0.4	0.7	0.8	0.9	1.1	1.2
0.9	0.9	0.5	0.5	0.4	0.4	-0.5	-0.7	-1.1	-1.5	-1.9	-2.2	-2.6	-2.6	-2.9	-2.8	-2.6	-2.2	-1.8	-1.6	-0.9	-0.7	-0.5	0.4	0.4	0.5	0.8	0.9	1.1	1.1	1.2
1.1	0.9	0.7	0.8	0.5	0.4	0.4	-0.5	-0.7	-0.8	-1.1	-1.2	-1.6	-1.6	-1.8	-1.5	-1.6	-1.2	-1.1	-0.8	-0.5	-0.5	0.4	0.4	0.5	0.7	0.9	0.9	1.1	1.1	1.2
1.1	0.9	0.8	0.8	0.7	0.5	0.4	0.4	-0.5	-0.7	-0.7	-0.8	-0.8	-0.9	-0.8	-0.8	-0.7	-0.7	-0.5	-0.4	0.4	0.4	0.5	0.7	0.8	0.9	1.1	1.1	1.2	1.2	1.2
1.1	1.1	0.9	0.9	0.8	0.7	0.5	0.4	0.4	0.4	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	0.4	0.4	0.5	0.5	0.8	0.8	0.9	1.1	1.1	1.2	1.2
1.2	1.1	1.1	1.1	0.9	0.8	0.8	0.7	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.7	0.8	0.9	0.9	1.1	1.1	1.1	1.2	1.2	1.2

Training Set 2: 81mm Mortar

2.5	2.6	2.8	2.8	2.9	3.1	3.1	3.2	3.2	3.1	3.2	3.2	3.2	3.3	3.5	3.5	3.5	3.6	3.6	3.6	3.6	3.5	3.3	3.3	3.2	3.1	2.8	2.6	2.5	2.4	2.2	
2.6	2.8	2.9	3.1	3.2	3.2	3.2	3.3	3.3	3.2	3.3	3.2	3.3	3.3	3.5	3.6	3.8	3.9	3.9	3.9	3.9	3.9	3.8	3.5	3.5	3.3	3.2	2.9	2.6	2.6	2.4	
2.9	3.1	3.2	3.3	3.3	3.3	3.3	3.3	3.3	3.2	3.2	3.2	3.2	3.3	3.5	3.8	3.9	4.1	4.3	4.2	4.3	4.2	4.1	3.9	3.8	3.5	3.3	3.2	2.9	2.6	2.5	
3.1	3.2	3.3	3.5	3.6	3.6	3.5	3.5	3.3	3.2	3.2	2.9	3.1	3.1	3.5	3.6	3.9	4.3	4.6	4.6	4.6	4.6	4.3	4.3	4.1	3.9	3.5	3.3	3.1	2.9	2.6	
3.3	3.5	3.5	3.8	3.8	3.8	3.8	3.6	3.3	3.1	2.6	2.4	2.4	2.4	2.9	3.3	3.8	4.3	4.8	4.9	5	4.9	4.9	4.6	4.5	4.2	3.9	3.8	3.3	3.2	2.8	
3.5	3.8	3.9	4.1	4.1	4.1	3.9	3.6	3.2	2.6	2.1	1.5	1.4	1.2	1.9	2.6	3.3	4.3	4.9	5.3	5.5	5.5	5.3	5	4.9	4.6	4.3	3.9	3.6	3.3	3.1	
3.8	3.9	4.2	4.3	4.3	4.3	4.2	3.6	3.1	2.1	1.1	0.4	-0.5	-0.7	0.4	1.4	2.6	4.1	5	5.6	5.9	6	5.9	5.6	5.3	5	4.6	4.3	3.9	3.6	3.3	
4.1	4.3	4.5	4.6	4.8	4.6	4.3	3.8	2.8	1.5	0.4	-1.4	-2.4	-2.9	-1.2	-0.8	1.4	3.3	4.9	5.7	6.3	6.5	6.5	6.2	5.7	5.5	4.9	4.6	4.2	3.9	3.3	
4.3	4.6	4.9	5.2	5.2	5	4.6	3.9	2.8	1.9	-1.2	-3.6	-4.9	-6.2	-4.3	-3.2	0.4	2.5	4.8	5.9	6.7	7	6.9	6.7	6.3	6	5.5	4.9	4.5	4.2	3.8	
4.6	5	5.2	5.6	5.6	5.6	5	4.3	2.8	0.8	-2.4	-5.6	-7.7	-9.7	-8.4	-6.5	-2.1	1.2	4.3	6	7.2	7.6	7.6	7.4	6.9	6.6	5.7	5.5	4.8	4.3	3.9	
4.9	5.3	5.6	6	6.3	6.3	5.9	5	3.3	0.8	-2.9	-7.4	-10.3	-12.8	-12.7	-9.1	-5.2	-0.5	3.9	6.2	7.7	8.4	8.3	8	7.6	7	6.2	5.7	5	4.6	4.2	
5.3	5.7	6.2	6.6	6.9	7	6.7	5.9	4.1	1.5	-3.2	-8.2	-12.4	-15.8	-16.2	-12	-7.2	-0.7	3.9	6.7	8.6	9	9	8.7	8.2	7.6	6.7	6	5.3	4.9	4.3	
5.5	6.2	6.5	7.2	7.6	7.9	7.7	7	5	2.5	-2.8	-8.3	-14.1	-18.3	-18.1	-14.4	-8.7	-0.9	4.2	7.4	5.4	10.1	10	9.4	8.7	8.2	7.2	6.3	5.6	5	4.5	
5.7	6.5	6.9	7.6	8.2	8.7	8.6	8.2	6.5	3.9	-1.5	-7.6	-14.5	-19.3	-19.3	-15.4	-8.7	-0.5	5.3	8.9	10.7	11.1	10.8	10.1	9.4	8.6	7.6	6.7	5.9	5.3	4.8	
6	6.7	7.2	7.9	8.7	9.4	9.4	9.3	8.3	6.2	0.4	-5.9	-13.4	-18.5	-18.6	-14.8	-7.6	0.8	6.9	10.3	12	12.3	11.7	10.8	10	9	7.9	7	6.2	5.6	4.9	
6.2	6.9	7.4	8.2	9	9.8	10	10	9.3	7	2.1	-3.6	-11.7	-17.3	-17.1	-13.2	-5.7	2.4	8.4	11.4	13.1	13.1	12.4	11.4	10.4	9.3	8.2	7.2	6.3	5.7	5	
6.3	7	7.7	8.3	9.1	9.8	10	10.1	9.4	6.7	2.4	-3.9	-11.3	-17.9	-17.6	-12.7	-6	3.1	9.1	12.1	13.5	13.5	12.8	11.7	10.7	9.1	8.3	7.3	6.5	5.7	5.2	
6.3	7	7.7	8.3	9	9.6	9.6	9.6	8.6	4.9	0.5	-6.7	-13.2	-23.1	-22	-17.6	-8.9	0.8	0.8	11.3	13	13.2	12.7	11.5	10.6	9.3	8.3	7.3	6.5	5.7	5.2	
6.3	6.9	7.6	8.2	8.9	9	8.9	7.7	7	1.6	-3.3	-11.8	-20.2	-32.8	-32	-27	-16.2	-4.1	4.1	8.7	11.3	12.3	11.8	11	10.1	9	8.2	7.2	6.5	5.7	5.2	
6.2	6.7	7.4	7.9	8.4	8.4	7.9	5.6	4.9	-1.6	-8.3	-19.2	-28.4	-36	-36	-34.6	-27	-10.7	-0.7	4.9	8.7	10.1	10.6	10	9.4	8.4	7.7	7	6.2	5.6	5	
6	6.6	7.2	7.4	7.7	7.6	6.7	4.2	2.5	-5	-13.2	-25.8	-34.7	-36	-36	-36	-34.9	-20.3	-7.9	0.4	4.9	7.9	8.6	8.9	8.6	7.7	7.2	6.6	6	5.5	5	
5.9	6.3	6.9	7	7.2	6.9	5.9	3.2	0.7	-6.7	-16.2	-28.8	-36	-36	-36	-36	-29.2	-15.4	-6.2	0.7	4.9	6.5	7	7.3	6.7	6.6	6.2	5.7	5.2	4.8		
5.6	6	6.6	6.6	6.7	6.3	5.5	2.6	0.4	-6.7	-16.6	-28.4	-36	-36	-36	-36	-34.6	-21.2	-11.2	-3.3	0.9	3.9	4.9	5.9	5.7	6	5.6	5.5	4.9	4.6		
5.5	5.7	6	6.2	6.2	5.9	4.9	2.2	0.4	-5.6	-14.2	-23.6	-35.7	-36	-36	-36	-36	-37.2	-17.3	-7.6	-2.2	1.5	3.2	4.6	4.9	5.2	5	4.9	4.6	4.3		
5.2	5.5	5.7	5.6	5.6	5	4.3	1.6	0.4	-4.6	-11.3	-18.1	-30.5	-36	-36	-36	-36	-30.6	-20.5	-10.7	-5	-0.7	1.5	3.3	4.1	4.6	4.5	4.5	4.3	4.2		
4.9	5	5.3	4.9	4.9	4.2	3.3	1.7	-0.5	-4.8	-9.4	-16.1	-26.4	-36	-36	-36	-36	-36	-33.3	-24.8	-13.5	-7.9	-2.5	0.4	2.1	3.3	3.9	4.1	4.2	3.9	3.9	
4.6	4.8	4.8	4.3	4.2	3.3	2.1	-1.1	-2.1	-7.6	-12.5	-20.5	-28.4	-36	-36	-36	-36	-36	-35.6	-28.8	-16.2	-9.8	-4.2	-0.9	1.1	2.5	3.2	3.6	3.8	3.6	3.6	
4.3	4.3	4.3	3.8	3.3	1.9	0.7	-3.5	-4.6	-12.3	-18.9	-30	-36	-36	-36	-36	-36	-36	-30.9	-18.2	-11.3	-5.6	-2.1	0.4	1.6	2.6	3.1	3.3	3.3	3.3		
3.3	3.9	3.8	3.1	2.6	1.9	-0.9	-5.9	-7.4	-18.9	-28.1	-36	-36	-36	-36	-36	-36	-36	-31.5	-19	-11.5	-6.2	-2.6	-0.5	1.4	2.2	2.6	3.1	3.2	3.2		
3.6	3.6	3.3	2.6	1.6	1.4	-1.9	-7.6	-10.7	-21.4	-32.6	-36	-36	-36	-36	-36	-36	-36	-30.2	-18.5	-11.1	-6.2	-2.8	-0.7	1.1	2.1	2.4	2.8	2.9	3.1		
3.3	3.3	2.9	2.2	1.1	-0.7	-2.9	-8	-12	-22.3	-33.3	-36	-36	-36	-36	-36	-36	-36	-33.7	-27.4	-16.8	-10	-5.6	-2.6	-0.7	0.9	1.8	2.2	2.6	2.6	2.8	
3.2	3.1	2.6	1.9	1.9	-0.8	-2.8	-7.6	-11.1	-20.3	-30.6	-36	-36	-36	-36	-36	-36	-36	-28.4	-22.7	-14.5	-8.7	-4.5	-2.1	-0.5	0.9	1.6	2.1	2.5	2.6	2.6	
2.9	2.8	2.4	1.9	1.8	-0.8	-2.5	-6.2	-9.3	-16.4	-25	-30.8	-36	-36	-36	-36	-36	-35.7	-31.9	-22	-17.8	-11.7	-7.3	-3.8	-1.6	-0.5	0.9	1.6	2.1	2.4	2.5	2.5
2.8	2.6	2.2	1.8	1.1	-0.5	-1.8	-4.5	-6.9	-12.3	-17.3	-22.6	-26.8	-29.4	-30.6	-30.8	-29.8	-22.3	-16.6	-13.2	-8.3	-5.5	-2.8	-1.1	-0.5	1.1	1.6	1.9	2.2	2.7	2.4	
2.6	2.6	2.2	1.8	1.2	-0.4	-0.9	-3.1	-4.6	-8.3	-11.4	-15.4	-18.9	-19.9	-20.7	-20	-17.6	-15.4	-11.4	-9.4	-6	-3.8	-1.8	-0.8	0.4	1.2	1.6	1.9	2.2	2.2	2.4	

Training Set 2: 105mm Artillery

6.6	4	4.3	4.8	5.2	5.7	6.2	6.9	7.4	8	8.4	8.7	8.7	9.3	9.1	9	8.7	8.3	8	7.7	7.2	6.7	6.2	5.7	5.2	4.8	3.9	3.9	3.5	3.2	3.1
3.9	4.3	4.9	5.3	5.9	6.6	7.2	8	8.7	9.3	9.8	10.1	10.4	11.1	11	10.8	10	10	9.6	9.3	8.3	7.7	6.9	6.6	5.7	5.3	4.3	4.3	3.9	3.5	3.3
4.3	4.8	5.3	6	6.7	7.3	8.3	9.3	10.1	11	11.4	12.1	12.4	13	12.8	12.8	11.8	12	11.1	10.7	9.4	8.9	7.9	7.7	6.5	6	4.9	4.8	4.2	3.9	3.5
4.6	5.2	5.7	6.6	7.3	8.3	9.1	10.4	11.4	12.7	13.2	14.4	14.5	15.4	15	15.1	13.9	13.9	12.8	12.3	10.8	10.3	9	8.4	7.2	6.6	5.2	5.2	4.5	4.2	3.8
4.9	5.5	6.3	7.2	8	9.1	10.3	11.7	13.4	14.2	15.5	16.5	17.3	18.1	17.8	17.5	16.8	16.6	14.8	14.1	12.5	11.8	10	9.4	8	7.2	5.6	5.6	4.9	4.3	3.9
5.2	5.9	6.9	7.7	8.7	10.1	11.4	13.2	14.9	16.8	17.9	19.2	20	20.7	20.3	20.5	19.3	18.9	17.2	16.2	14.4	13.1	11.3	10.4	8.9	7.9	6.2	6	5.3	4.8	4.2
5.5	6.2	7.3	8.3	9.4	11	12.8	14.7	16.6	18.6	20	21.7	22.4	23.4	23	22.7	22	21.6	19.6	18.1	16.1	14.7	12.5	11.4	9.6	8.6	6.6	6.5	5.6	4.9	4.3
5.6	6.5	7.6	8.7	10	11.7	13.7	15.8	18.1	20.2	21.9	23.4	24.1	24.1	24.8	24.7	24.3	24	21.4	20.2	17.9	15.9	13.7	12.4	10.3	9.1	7	6.7	5.9	5.2	4.6
5.9	6.7	7.9	8.9	10.4	12.1	14.1	16.5	18.6	20.9	22.6	23.6	24.3	23.8	24.7	24.3	24.8	24.8	23.3	21.4	19.2	17.2	14.7	13.2	11	9.7	7.4	7	6.2	5.5	4.8
5.9	6.7	8	9	10.4	12.1	14.2	16.5	18.6	20.3	21.6	20.9	20.2	17	18.3	20.7	22.9	23.3	23.6	22	20.2	18.1	15.2	13.7	11.4	10.8	7.7	7.2	6.3	5.6	4.9
5.9	6.6	7.9	8.8	10.1	11.7	13.5	15.2	17.2	18.5	18.6	14.4	13.7	3.9	10.6	13	18.3	19.7	22.4	21.6	20.2	18.2	15.6	13.9	11.7	10.3	8	7.3	6.5	5.7	5
5.7	6.5	7.6	8.4	9.6	11	12.4	13.7	14.5	14.4	12.4	14.2	0.4	-6.7	-8.4	-0.5	8.9	12.8	19.6	19.9	19.2	17.6	15.5	13.7	10.7	10.3	8.3	7.4	6.6	5.7	5.2
5.5	6.2	7	7.7	8.7	9.6	10.6	11	10.1	9	5.5	-7.9	-13.8	-24.4	-29.2	-15.2	-5	4.2	13.9	15.9	16.9	16.2	14.7	13	10.7	10	8.3	7.3	6.6	5.7	5.2
5.2	5.7	6.5	7	7.4	7.9	7.7	7.2	4.9	0.5	-6.5	-20.7	-32.2	-36	-36	-36	-23.3	-7.7	5	11.1	13.4	13.8	13.1	11.4	10.1	9.3	8	7.2	6.5	5.6	5
4.9	5.2	5.6	5.6	6	5.6	4.6	2.9	-1.6	-8.7	-17.3	-33.9	-36	-36	-36	-36	-36	-20.5	-5.3	3.8	8.7	10.3	11	10.1	9.1	8.7	7.7	6.9	6.3	5.5	4.9
4.3	4.3	4.8	4.6	4.5	3.5	1.6	-1.2	-9	-16.5	-29.5	-36	-36	-36	-36	-36	-36	-33	-15.8	-3.3	3.2	6.2	8.4	8.2	8.3	7.7	7.2	6.5	6	5.3	4.9
3.9	3.9	3.9	3.5	2.9	1.6	-1.9	-5.6	-14.7	-23.6	-36	-36	-36	-36	-36	-36	-36	-36	-26.3	-11.1	-2.1	2.1	5.6	6.3	7.3	6.9	6.6	6	5.6	5	4.6
3.6	3.3	3.2	2.4	1.6	-0.4	-4.3	-11	-19	-29.8	-36	-36	-36	-36	-36	-36	-36	-36	-34.2	-17.5	-7.7	-1.2	3.2	4.5	6	5.7	5.9	5.6	5.3	4.8	4.5
3.3	3.1	2.6	1.6	0.4	-1.6	-6	-12	-26	-33.3	-36	-36	-36	-36	-36	-36	-36	-36	-36	-24.8	-10.7	-5.6	0.4	2.2	4.9	4.8	5.2	4.9	4.9	4.5	4.2
2.9	2.6	2	1.2	-0.7	-2.8	-7.3	-14.5	-23.3	-34.7	-36	-36	-36	-36	-36	-36	-36	-36	-36	-28.8	-14.5	-7.7	-1.5	0.8	3.9	3.9	4.6	4.5	4.5	4.1	3.9
2.6	2.2	1.5	0.7	-0.9	-3.5	-8	-14.8	-23.4	-34.5	-36	-36	-36	-36	-36	-36	-36	-36	-36	-29.8	-15.1	-9	-3.2	-0.5	3.1	3.1	4.1	3.9	4.1	3.9	3.8
2.4	2.1	1.2	0.4	-1.2	-3.5	-7.9	-13.9	-22.6	-32	-36	-36	-36	-36	-36	-36	-36	-36	-36	-28.2	-14.9	-9.4	-4.2	-0.8	2.4	2.5	3.5	3.5	3.6	3.5	3.5
2.2	1.8	1.1	0.4	-1.2	-3.3	-7.2	-12.3	-19	-27.7	-36	-36	-36	-36	-36	-36	-36	-36	-36	-25	-13.7	-9	-4.3	-0.9	1.9	2.1	3.1	3.2	3.3	3.3	3.2
2.1	1.6	1.1	0.4	-0.9	-2.8	-5.7	-9.8	-15.5	-21.4	-31.1	-35.4	-36	-36	-36	-36	-36	-36	-31.1	-20.2	-11	-7.7	-3.9	-0.9	1.6	1.9	2.8	2.9	3.1	3.2	3.1
2.1	1.6	1.1	0.4	-0.7	-2.1	-4.8	-7.9	-12.4	-16.2	-23.7	-27.1	-34.6	-36	-36	-36	-34.5	-26.4	-22.4	-14.2	-8.6	-5.7	-2.6	-0.7	1.6	1.9	2.6	2.8	2.9	2.9	2.9
1.9	1.6	1.1	0.7	-0.5	-1.5	-3.5	-5.7	-8.9	-12	-16.8	-18.9	-25.5	-27	-29.4	-26.5	-23	-18.1	-15.2	-11	-6.5	-4.2	-1.9	-0.5	1.6	1.8	2.4	2.6	2.8	2.8	2.8
1.9	1.6	1.2	0.9	0.4	-0.7	-2.4	-3.9	-6.5	-8.2	-11	-12.3	-16.1	-17.9	-18.9	-16.4	-16	-12.1	-9	-7.3	-4.2	-2.6	-1.2	0.4	1.6	1.8	2.4	2.5	2.6	2.6	2.6
1.9	1.6	1.4	1.1	0.4	-0.5	-1.2	-2.2	-4.1	-5.2	-7.2	-8.2	-10.3	-10.7	-12.4	-10.7	-9.8	-7.2	-6	-4.6	-2.5	-1.4	-0.5	0.8	1.6	1.9	2.2	2.4	2.5	2.5	2.5
1.9	1.8	1.5	1.2	0.8	0.4	-0.5	-1.2	-2.4	-2.9	-4.3	-4.6	-6.2	-6.3	-7.3	-6.7	-5.9	-4.9	-3.5	-2.5	-1.2	-2.7	0.4	0.9	1.8	1.9	2.2	2.4	2.5	2.5	2.4
1.9	1.8	1.6	1.5	1.1	0.8	0.4	-0.5	-1	-1.6	-2.4	-2.6	-3.8	-3.9	-4.3	-3.5	-3.3	-2.4	-1.6	-1.1	-0.5	0.4	0.8	1.4	1.9	2.1	2.2	2.4	2.4	2.4	2.4
1.9	1.9	1.7	1.6	1.4	1.1	0.7	0.4	-0.5	-0.5	-1.1	-1.1	-1.9	-2.2	-2.1	-1.8	-1.6	-0.9	-0.7	-0.5	0.4	0.8	1.2	1.6	1.9	2.1	2.2	2.4	2.4	2.4	2.4
1.9	1.9	1.8	1.6	1.5	1.4	1.1	0.8	0.4	0.4	-0.5	-0.5	-0.7	-0.8	-1	-0.7	-0.7	-0.4	0.4	0.5	0.9	1.2	1.5	1.8	2.1	2.1	2.2	2.2	2.2	2.2	2.2
1.9	1.9	1.9	1.8	1.6	1.5	1.4	1.2	0.9	0.8	0.5	0.4	0.4	-0.5	-0.5	-0.4	0.4	0.5	0.8	0.9	1.2	1.5	1.6	1.9	2.1	2.1	2.2	2.2	2.2	2.2	2.2
1.9	1.9	1.9	1.8	1.8	1.6	1.5	1.5	1.2	1.2	0.9	0.8	0.5	0.4	0.4	0.4	0.5	0.9	1.1	1.4	1.5	1.6	1.8	1.9	2.1	2.1	2.2	2.2	2.2	2.1	2.1
1.9	1.9	1.9	1.9	1.8	1.8	1.6	1.6	1.5	1.4	1.2	1.1	0.9	0.8	0.8	0.8	0.9	1.2	1.4	1.6	1.6	1.9	1.9	2.1	2.1	2.1	2.1	2.1	2.1	2.1	2.1

Training Set 2: 105mm HEAT

3.2	3.6	4.2	4.8	5.5	6.5	7.7	9.3	10.6	12.4	14.7	17.2	19.5	21.4	21.7	22.6	20.9	19.3	17.8	14.9	13	11.1	9.4	8.3	6.9	5.9	5	4.3	3.9	3.5	2.9
3.3	3.9	4.3	5	5.7	6.9	8.3	10	11.8	14.1	17.3	20.5	23	25.5	26.1	27.1	25.3	22.9	20.7	17.5	14.9	12.7	10.7	9	7.6	6.3	5.5	4.6	4.1	3.6	3.1
3.5	3.9	4.5	5.2	6	7.3	8.7	10.6	12.7	15.2	8.9	22.3	25.7	28.8	29.8	30.9	29.2	26.7	23.6	19.9	16.8	13.9	11.4	9.8	8.3	6.9	5.7	4.9	4.3	3.8	3.3
3.6	4.1	4.6	5.5	6.2	7.4	9	11	13.1	15.9	19.7	23.6	27.5	30.5	32.2	32.6	31.5	28.7	26	21.4	17.9	15.1	12.3	10.6	8.7	7.2	6	5.2	4.5	3.9	3.3
3.6	4.2	4.8	5.5	6.3	7.6	8.9	11	13.1	15.9	19.2	23	27.1	29.6	31.8	32.2	31.2	29.2	26.7	22.2	18.3	15.5	12.7	10.8	9	7.6	6.2	5.5	4.6	4.2	3.5
3.6	4.1	4.6	5.5	6.2	7.3	8.4	10.4	12.1	14.8	17.6	20	24	24.8	28.5	28.8	28.4	27	25.5	21.9	18.3	15.5	12.8	10.8	9.1	7.6	6.3	5.5	4.8	4.2	3.5
3.5	4.1	4.6	5.2	5.7	6.9	7.9	9.4	10.6	12.4	13.5	13.7	15.2	14.8	19.3	19.9	22.9	22.7	22.4	19.9	17.5	15.1	12.5	10.7	9	7.6	6.5	5.5	4.8	4.3	3.6
3.5	3.9	4.3	4.9	5.5	6.2	7.2	7.9	8.2	9.4	9.3	8.2	6	2.4	5.5	6.6	13.5	15.8	17.3	17.5	13.9	13.9	12	10.3	9	7.6	6.3	5.5	4.8	4.3	3.6
3.3	3.8	4.2	4.5	4.8	5.5	5.7	5.6	5.6	4.9	2.8	-0.7	-5	-13.5	-6.9	-6.5	1.6	6.9	12	13.1	13.5	12.3	11.1	9.6	8.4	7.2	6.2	5.5	4.8	4.2	3.6
3.3	3.6	3.9	4.1	4.2	4.3	4.2	3.1	2.8	0.4	-3	-8.9	-17.5	-26	-27	-21.9	-10.3	-2.1	9.3	7.9	10.7	10	9.6	8.7	7.9	6.7	6	5.3	4.6	4.2	3.6
3.1	3.3	3.5	3.6	3.3	3.3	2.8	1.4	-0.7	-3.6	-9.4	-18.1	-28	-35.5	-36	-35.6	-23.1	-13	-4.3	3.6	7.6	7.6	8	7.6	7.3	6.3	5.6	5	4.5	4.1	3.6
2.9	3.1	3.2	3.3	2.8	2.4	1.2	-0.5	-3.9	-8.7	-16.2	-26.7	-36	-36	-36	-36	-35	-23.7	-12	-1.9	3.3	5	6.5	6.2	6.5	5.7	5.3	4.8	4.3	3.9	3.5
2.8	2.9	2.9	2.8	2.4	1.5	0.4	-2.1	-6.3	-12.1	-21.3	-33.3	-36	-36	-36	-36	-36	-31.8	-19.3	-6.7	-0.5	2.8	4.8	5.3	5.7	5.3	5	4.5	4.2	3.8	3.5
2.8	2.6	2.6	2.4	1.6	0.9	-0.7	-3.3	-8.2	-14.8	-24.8	-35.4	-36	-36	-36	-36	-36	-25	-10.8	-3.3	0.7	3.5	4.2	4.9	4.6	4.6	4.3	4.1	3.6	3.3	
2.6	2.5	2.6	2.2	1.5	0.4	-1.1	-3.8	-8.9	-16.8	-25.8	-35.6	-36	-36	-36	-36	-36	-29.4	-14.7	-5.7	-0.8	2.1	3.3	4.3	4.3	4.2	4.1	3.9	3.5	3.3	
2.6	2.4	2.4	2.1	1.2	0.4	-1.1	-3.8	-8.7	-15.5	-23.6	-34.5	-36	-36	-36	-36	-36	-29.8	-15.1	-6.7	-1.6	1.5	2.8	3.8	3.9	3.9	3.8	3.8	3.3	3.2	
2.5	2.4	2.2	1.9	1.1	0.4	-0.8	-3.6	-7.3	-13.5	-20	-30.6	-36	-36	-36	-36	-36	-34.5	-26.4	-13.8	-6.3	-1.6	1.2	2.6	3.3	3.6	3.8	3.6	3.5	3.3	3.1
2.5	2.4	2.2	1.9	1.1	0.7	-0.7	-2.8	-6	-10.1	-16.2	-23.6	-32.6	-36	-36	-36	-35.7	-26	-20.9	-10.3	-4.3	-0.7	1.5	2.6	3.3	3.5	3.6	3.5	3.3	3.2	3.1
2.4	2.2	2.1	1.9	1.2	0.9	-0.5	-1.6	-4.2	-6.6	-11.4	-15.8	-22.9	-23.8	-30.4	-29.2	-25.3	-14.8	-13.9	-5.6	-2.1	0.5	1.9	2.9	3.3	3.5	3.5	3.3	3.3	3.1	2.9
2.2	2.2	2.1	1.9	1.4	1.2	0.4	-0.7	-2.1	-3.9	-7.4	-10	-13.2	-13.8	-16.2	-13	-12.4	-6.2	-6.2	-1.2	0.4	0.9	2.6	3.2	3.3	3.5	3.3	3.3	3.2	3.1	2.8
2.2	2.1	2.1	1.9	1.5	1.2	0.5	0.4	-1.1	-2.2	-4.1	-5	-7.3	-6	-7.7	-4.3	-3.9	-0.5	-0.5	1.5	2.4	2.9	3.3	3.5	3.5	3.3	3.3	3.2	3.1	2.8	2.8
2.1	2.1	1.9	1.8	1.4	1.2	0.8	0.4	-0.7	-1.6	-2.6	-3.5	-4.5	-4.9	-4.1	-2.4	-0.5	1.5	2.5	3.2	3.5	3.5	2.5	3.5	3.5	3.3	3.2	3.1	2.9	2.8	2.8
2.1	1.9	1.8	1.6	1.2	1.1	0.4	-0.4	-0.8	-2.1	-3.1	-4.5	-5.7	-7.9	-6.3	-5.3	-1.5	-0.5	1.9	2.8	2.3	3.3	2.3	3.3	3.3	3.2	3.1	2.9	2.8	2.6	2.6
1.9	1.9	1.6	1.5	0.9	0.5	0.4	-0.8	-1.5	-3.3	-5.2	-7.7	-9.3	-14.4	-11.4	-11.4	-6.7	-4.8	-0.8	0.8	2.2	2.6	2.9	2.8	2.7	2.8	2.8	2.6	2.6	2.5	2.6
1.8	1.6	1.6	1.4	0.8	0.4	-0.5	-1.6	-3.1	-5.6	-8	-11.4	-15.2	-20.6	-19.7	-20.9	-12.8	-9.6	-5.2	-1.5	0.4	1.2	1.9	2.2	1.5	2.6	2.6	2.5	2.5	2.4	2.5
1.6	1.6	1.4	1.1	0.4	-0.5	-1	-2.5	-4.2	-7.6	-10.7	-14.7	-21.6	-28.1	-27.9	-27.4	-20.7	-15.2	-9.8	-4.9	-1.6	-0.5	0.7	1.5	1.1	2.1	2.2	2.2	2.4	2.2	2.4
1.6	1.5	1.2	0.9	0.4	-0.7	-1.6	-3.3	-3.5	-9.7	-13.7	-17.5	-25.7	-32.6	-33.5	-33.3	-27	-20.7	-13.9	-8	-3.9	-1.5	-0.5	0.8	1.5	1.6	1.9	2.1	2.1	2.1	2.2
1.6	1.4	1.1	0.7	-0.5	-0.9	-2.2	-3.9	-6.9	-11	-15.2	-19.2	-29.9	-35	-36	-36	-31.2	-25.1	-18.1	-10.7	-6.2	-3.1	-1.1	-0.4	0.9	1.2	1.6	1.8	1.9	1.9	2.1
1.5	1.2	0.9	0.5	-0.5	-1.2	-2.5	-4.3	-7	-12	-16.8	-20.6	-30.4	-35.6	-36	-36	-33	-26.7	-20.2	-12.4	-7.4	-4.1	-1.9	-0.7	0.4	0.9	1.4	1.6	1.8	1.8	1.9
1.5	1.1	0.8	0.4	-0.5	-1.4	-2.8	-4.3	-7.6	-12	-16.9	-20.7	-29.8	-34.7	-36	-35.6	-32.8	-26.3	-20.2	-12.8	-8.2	-4.5	-2.4	-0.8	-0.4	0.7	1.1	1.6	1.6	1.6	1.8
1.4	1.1	0.8	0.4	-0.5	-1.4	-2.5	-4.2	-7.2	-11	-15.4	-19.2	-26.7	-31.6	-34.5	-32.8	-30.6	-24	-19	-12.3	-8	-4.6	-2.6	-0.9	-0.5	0.4	0.9	1.2	1.5	1.6	1.6
1.2	1.1	0.8	0.4	-0.5	-1.2	-2.4	-3.9	-6.3	-9.6	-13.4	-15.9	-22.4	-26.8	-31.2	-28.1	-25.5	-20.7	-15.2	-10.7	-7.4	-4.3	-2.6	-1.1	-0.5	0.4	0.8	1.1	1.4	1.5	1.6
1.2	1.1	0.8	0.4	-0.5	-1.1	-2.1	-3.2	-5.5	-7.9	-11	-12.1	-18.1	-28.2	-23.3	-21.4	-19.7	-17.5	-13.2	-9.1	-6.6	-3.8	-2.6	-0.9	-0.5	0.4	0.8	1.1	1.2	1.4	1.6
1.2	1.1	0.8	0.4	-0.5	-0.8	-1.5	-2.5	-4.2	-6	-8.4	-9.6	-13	-15.2	-16.6	-16.1	-14.4	-13.4	-10.3	-7.3	-5.3	-3.3	-2.1	-0.8	-0.5	0.4	0.7	0.9	1.2	1.4	1.5
1.2	1.1	0.8	0.7	0.4	-0.5	-1.1	-1.8	-3.2	-4.5	-5.9	-7.3	-9.3	-10.1	-11	-11.1	-10.1	-9.4	-7.9	-5.6	-4.1	-2.6	-1.6	-0.7	-0.5	0.4	0.8	0.9	1.2	1.4	1.5

Test Set: 3-5in Rocket

1.6	1.8	1.9	2.1	2.2	2.6	2.8	3.1	3.2	3.3	3.8	3.9	3.8	3.9	3.9	3.8	3.6	3.3	3.2	3.1	2.8	2.6	2.5	2.2	2.1	1.9	1.9	1.6	1.5	1.5	1.4
1.6	1.9	2.1	2.4	2.5	2.8	3.2	3.3	3.8	4.1	4.5	4.6	4.8	4.8	4.8	4.6	4.5	4.1	3.8	3.5	3.3	2.9	2.8	2.5	2.2	2.1	1.9	1.6	1.6	1.5	1.5
1.8	2.1	2.2	2.6	2.8	3.2	3.5	3.9	4.3	4.8	5.2	5.7	5.6	5.9	5.6	5.7	5.2	5.2	4.3	4.2	3.8	3.3	3.1	2.8	2.4	2.2	1.9	1.9	1.6	1.6	1.5
1.8	2.2	2.4	2.8	3.1	3.5	3.9	4.5	5.2	5.6	6.5	7	6.9	7.2	7.2	6.7	6.6	6.3	5	5	4.3	3.8	3.3	3.1	2.6	2.4	2.1	1.9	1.8	1.6	1.5
1.9	2.4	2.6	3.1	3.3	3.9	4.5	5.2	6.2	6.9	7.7	8.4	8.6	8.9	8.9	8.4	7.9	7.4	6.6	5.9	5	4.3	3.8	3.3	2.9	2.6	2.2	2.1	1.9	1.6	1.5
2.1	2.4	2.8	3.2	3.8	4.5	5	5.9	7.2	8	9.4	10.6	10.1	11.1	11	11.1	9.7	9	7.7	6.7	5.7	4.9	4.2	3.8	3.2	2.6	2.4	2.2	1.9	1.8	1.6
2.1	2.5	2.9	3.5	3.9	4.9	5.6	6.7	8	9.4	11.3	12.7	13	14.1	13.7	13.5	11.7	10.7	9.1	7.9	6.6	5.5	4.6	3.9	3.3	2.9	2.5	2.2	2.1	1.8	1.6
2.1	2.6	2.9	3.5	4.1	5.2	5.9	7.4	8.9	10.7	12.8	15.2	15.2	17.3	16.2	16.5	13.9	13	10.4	9	7.4	6.2	5	4.3	3.5	3.1	2.6	2.4	2.1	1.9	1.6
2.1	2.6	3.1	3.5	4.2	5.3	6.3	7.7	9.7	11.7	14.9	17.6	18.1	20.6	21	19.6	16.9	14.9	12.3	10	8.3	6.6	5.5	4.5	3.8	3.2	2.6	2.4	2.1	1.9	1.6
2.1	2.6	2.9	3.5	4.1	5.3	6.2	7.9	10	12.4	15.9	18.9	20.6	23.3	23	22	18.6	16.9	13.4	11	8.9	7	5.6	4.6	3.9	3.3	2.8	2.5	2.2	1.9	1.7
2.1	2.4	2.8	3.3	3.9	4.9	5.7	7.4	9.4	11.5	15.5	18.3	21.2	23.4	24.3	23.3	20.5	17.5	14.2	11.3	9.1	7.2	5.9	4.8	3.9	3.3	2.9	2.5	2.2	1.9	1.8
1.9	2.2	2.5	3.1	3.5	4.3	5	6.2	7.9	9.7	13.2	14.9	19.2	20.5	22.9	21.6	20.2	16.2	13.9	10.8	9	6.9	5.7	4.6	3.9	3.2	2.8	2.4	2.2	1.9	1.6
1.8	2.1	2.2	2.5	2.4	3.5	3.9	4.5	5.6	6	8.3	9	13.8	13.5	15.9	16.4	16.6	13.2	12.4	9.8	8.4	6.3	5.5	4.3	3.8	3.1	2.6	2.4	2.1	1.8	1.6
1.6	1.8	1.8	2.1	2.1	2.1	2.5	1.9	1.8	1.2	2.1	-0.8	-1.2	1.9	5.3	6.3	9.7	8.6	10.1	7.4	7	5.3	4.8	3.9	3.3	2.8	2.6	2.2	2.1	1.8	1.6
1.4	1.5	1.4	1.4	1.2	0.5	0.8	-1.2	-2.4	-3.2	-7.3	-13.5	-9.6	-12.7	-7.2	-4.1	4.2	3.9	6.2	4.9	5	3.9	4.2	3.3	3.1	2.6	2.4	2.1	1.9	1.6	1.6
1.1	1.2	0.9	0.8	0.5	-0.8	-1.1	-3.9	-5.2	-7.9	-14.5	-20.7	-22.3	-27.7	-20.6	-14.9	-7.2	-3.9	0.8	1.5	2.8	2.8	3.1	2.8	2.6	2.2	2.2	1.9	1.9	1.6	1.6
0.9	1.1	0.5	0.4	-0.5	-1.8	-2.6	-5.7	-8.3	-13.8	-20.5	-28.4	-31.6	-33.7	-29.6	-23.3	-14.7	-9	-3.3	-1.1	0.4	1.1	1.9	1.9	2.1	1.9	1.9	1.8	1.6	1.5	1.6
0.8	1.8	0.4	-0.4	-0.8	-2.4	-3.9	-7.2	-11	-16.9	-24.3	-31.8	-35	-36	-34.3	-29	-19.3	-12.4	-6.9	-3.3	-1.2	0.4	0.9	1.4	1.6	1.6	1.6	1.5	1.6	1.4	1.5
0.7	0.5	0.4	-0.5	-1.2	-3.1	-4.5	-7.9	-11.8	-17.9	-26.5	-32.6	-35.7	-36	-36	-30.9	-23.1	-15.4	-9.6	-5.9	-2.9	-1.1	0.4	0.5	1.1	1.2	1.5	1.4	1.5	1.2	1.5
0.5	0.4	-0.4	-0.7	-1.5	-3.3	-5	-7.4	-12.4	-17.5	-27	-31.5	-35.2	-36	-35.9	-30.6	-24.3	-16.2	-11	-6.9	-3.9	-1.6	-0.7	0.4	0.8	0.9	1.2	1.2	1.4	1.2	1.4
0.4	0.4	-0.5	-0.7	-1.6	-3.3	-4.9	-7.3	-11.7	-15.6	-23.7	-27.5	-32.6	-33.6	-33.9	-27.7	-22.9	-15.8	-11.1	-7.2	-4.3	-2.2	-0.9	-0.5	0.5	0.8	0.9	1.1	1.2	1.2	1.2
0.4	0.4	-0.5	-0.7	-1.6	-2.9	-4.5	-6.5	-10.4	-13.2	-19.6	-22.6	-27.5	-28.5	-29.6	-22.7	-19.9	-14.1	-10.3	-6.7	-4.5	-2.4	-1.1	-0.5	0.4	0.5	0.8	0.9	1.1	1.1	1.2
0.4	0.4	-0.5	-0.7	-1.4	-2.6	-4.2	-5.5	-8.7	-11	-15.8	-17.6	-22	-21.3	-23	-17.1	-15.8	-11.3	-8.9	-6	-4.2	-2.2	-1.2	-0.5	0.4	0.5	0.8	0.9	1.1	1.1	1.2
0.4	0.4	-0.5	-0.7	-1.2	-2.1	-3.3	-4.3	-6.7	-8	-12.1	-12.3	-16.2	-16.1	-16.4	-13.2	-12.3	-8.9	-7.3	-4.9	-3.6	-1.9	-1.1	-0.5	0.4	0.5	0.8	0.9	0.9	1.1	1.1
0.5	0.4	-0.4	-0.5	-0.8	-1.6	-2.6	-3.3	-5	-6.5	-9.3	-9.3	-11	-11.4	-12	-9.6	-9	-6.7	-5.5	-3.9	-2.8	-1.5	-0.9	-0.5	0.4	0.5	0.8	0.9	0.9	1.1	1.1
0.5	0.5	0.4	-0.4	-0.7	-1.1	-2.1	-2.2	-3.9	-4.5	-7	-6.7	-8.4	-8.3	-8.7	-6.9	-6.6	-4.9	-4.1	-2.9	-2.1	-1.1	-0.7	-0.5	0.4	0.5	0.8	0.9	0.9	1.1	1.1
0.5	0.5	0.4	0.4	-0.5	-0.7	-1.4	-1.6	-2.8	-3.3	-4.6	-4.8	-5.7	-5.5	-6	-4.8	-4.5	-3.5	-2.9	-2.1	-1.5	-0.8	-0.5	0.4	0.4	0.7	0.8	0.9	0.9	1.1	1.1
0.5	0.7	0.4	0.4	-0.5	-0.5	-0.8	-0.9	-1.8	-2.1	-3.1	-3.1	-3.5	-3.5	-3.8	-3.2	-3.2	-2.5	-2.1	-1.4	-0.9	-0.5	-0.5	0.4	0.4	0.8	0.8	0.9	0.9	1.1	1.1
0.7	0.8	0.5	0.5	0.4	0.4	-0.5	-0.7	-1.1	-1.2	-1.8	-1.8	-2.6	-2.2	-2.5	-1.9	-2.1	-1.5	-1.2	-0.8	-0.7	-0.5	0.4	0.4	0.5	0.8	0.8	0.9	0.9	1.1	1.1
0.8	0.8	0.5	0.7	0.4	0.4	-0.5	-0.5	-0.7	-0.7	-0.9	-0.9	-1.4	-1.4	-1.5	-1.1	-1.1	-0.8	-0.8	-0.5	-0.5	0.4	0.4	0.5	0.7	0.8	0.9	0.9	0.9	1.1	1.1
0.8	0.9	0.7	0.8	0.5	0.5	0.4	0.4	-0.5	-0.5	-0.7	-0.5	-0.7	-0.7	-0.8	-0.5	-0.7	-0.5	-0.5	0.4	0.4	0.4	0.4	0.7	0.8	0.8	0.9	0.9	1.1	1.1	1.1
0.9	0.9	0.8	0.9	0.7	0.7	0.4	0.4	0.4	0.4	-0.4	-0.5	-0.5	-0.5	-0.5	-0.5	-0.4	0.4	0.4	0.4	0.5	0.7	0.8	0.9	0.9	0.9	1	1.1	1.1	1.1	1.1
0.9	0.9	0.8	0.9	0.8	0.8	0.7	0.7	0.5	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.5	0.7	0.8	0.8	0.9	0.9	0.9	1.1	1.1	1.1	1.1	1.1
1.1	1.1	0.8	1.1	0.8	0.9	0.8	0.8	0.7	0.7	0.5	0.5	0.4	0.4	0.4	0.4	0.4	0.5	0.5	0.7	0.7	0.8	0.8	0.9	0.9	0.9	1.1	1.1	1	1.1	1.1
1.1	1.1	0.9	1.1	0.9	1.1	0.9	0.9	0.8	0.8	0.8	0.8	0.5	0.7	0.7	0.7	0.7	0.7	0.8	0.8	0.8	0.8	0.9	0.9	0.9	1.1	1.1	1.1	1.1	1.1	1.1

Test Set: 60mm Mortar

2.5	2.6	3.1	3.3	3.9	4.2	5	5.6	6.2	6.6	7	7.6	7.9	8	8.7	8.2	7.6	7	6.7	6.2	5.7	5.2	4.9	4.2	3.9	3.3	3.2	2.8	2.6	2.4	2.2
2.6	3.1	3.3	3.8	4.2	4.6	5.7	6.2	7.2	7.7	8.6	9.4	9.8	10	10.7	10	9.4	8.7	8.3	7.4	6.7	5.9	5.5	4.8	4.3	3.8	3.3	2.9	2.8	2.5	2.2
2.8	3.2	3.5	4.2	4.8	5.3	6.6	7.2	8.4	9.4	10.1	11.7	12.3	12.5	13.5	13	11.3	10.7	10	8.7	8	7	6.5	5.5	4.9	4.2	3.8	3.3	3.1	2.6	2.4
3.1	3.5	3.9	4.6	5	6	7.3	8.4	9.7	11.4	12.3	15.2	15.5	16.4	17.1	15.8	13.9	13.1	12	10.7	9.4	8.4	7.2	6.2	5.5	4.6	4.2	3.5	3.3	2.8	2.5
3.2	3.6	4.1	4.9	5.5	6.7	8.2	9.6	11.1	13.2	14.9	17.9	19.2	20.2	22	19.9	17.8	16.4	14.4	12.7	11	9.4	8.3	7	6.2	5.2	4.3	3.9	3.5	3.1	2.6
3.3	3.8	4.3	5	6	7.3	9	10.6	12.3	14.8	17.6	21.4	24	25.3	26.7	24.8	22.4	20.6	17.3	15.1	13	10.8	9.3	7.9	6.6	5.6	4.8	4.1	3.6	3.2	2.8
3.3	3.9	4.5	5.3	6.3	7.7	9.4	11.4	13.5	16.8	20.7	24.7	28.8	30.8	32	30.6	27.2	24.3	20.9	17.5	14.5	12.1	10.3	8.7	7.3	6	5	4.3	3.8	3.3	2.9
3.3	3.9	4.5	5.5	6.5	7.9	9.8	11.8	14.2	17.8	22.2	26.5	31.2	33.6	35.2	34	31.1	27.9	23.7	19.7	16.6	13.7	11.3	9.4	7.9	6.3	5.3	4.6	3.9	3.5	3.1
3.5	4.1	4.5	5.5	6.3	7.9	9.6	11.7	14.5	17.6	22.2	26.5	31.8	33.6	35.4	35	33.5	30.5	26.1	21.6	17.9	14.7	12.1	9.8	8.2	6.6	5.5	4.8	4.1	3.5	3.1
3.3	3.9	4.3	5.3	6.2	7.7	9	11.1	13.4	16.4	19.6	23	28.2	30.4	21.2	32.6	33.2	30.5	26.8	22.4	18.9	15.2	12.7	10.1	8.4	6.7	5.6	4.9	4.2	3.6	3.1
3.2	3.8	4.2	4.9	5.5	7	7.9	9.8	11.3	13.1	14.5	13.7	18.2	20.3	20	23.3	29.5	27.1	25.8	21.9	19	15.1	12.8	10	8.4	6.9	5.6	4.9	4.2	3.6	3.2
3.1	3.5	3.9	4.5	4.9	6	7	7.7	8.4	9.3	8.4	6.2	5.9	4	3.9	8.3	19.7	19.5	21.9	19.6	17.5	14.1	12	9.6	8.3	6.6	5.6	4.8	4.2	3.5	3.2
2.8	3.2	3.3	3.9	4.2	4.6	4.8	5.6	4.2	4.6	0.5	-2.4	-6.7	-13	-11.7	-6.8	6.7	10.4	16.9	16.1	14.7	12.7	11	8.9	7.6	6.2	5.3	4.5	4.1	3.5	3.1
2.5	3.1	3.1	3.3	3.3	3.3	2.9	2.6	0.8	-0.7	-8.9	-13.2	-21.6	-30.4	-28.5	-20.7	-7.2	0.9	8.7	9.8	11.1	10	9.3	7.7	6.9	5.7	4.9	4.3	3.9	3.3	3.1
2.2	2.6	2.4	2.6	2.1	2.2	0.4	0.4	-2.4	-6.7	-16.4	-20.9	-23.5	-36	-36	-34	-20	-9.3	0.4	4.2	6.2	6.9	7	6.5	5.9	5	4.5	3.9	3.6	3.2	2.8
1.9	2.2	1.8	2.1	0.9	0.8	-1.2	-2.5	-6	-11.3	-22.9	-28.1	-36	-36	-36	-36	-30.9	-19.2	-8.9	-1.4	2.2	4.1	5	4.9	4.8	4.3	4.1	3.5	3.3	2.9	2.8
1.6	1.9	1.5	1.5	0.4	-0.5	-3.1	-4.8	-9.4	-15.1	-27.5	-30	-36	-36	-36	-36	-36	-29.4	-15.4	-8.2	-0.9	0.4	3.3	2.8	3.5	3.5	3.5	3.1	3.1	2.6	2.6
1.5	1.6	1.1	0.9	-0.7	-1.1	-4.2	-6.7	-10.6	-17.5	-29.1	-36	-36	-36	-36	-36	-36	-32.8	-22	-11.7	-5.7	-1.8	0.5	1.6	2.6	2.6	2.9	2.6	2.8	2.5	2.4
1.4	1.4	0.8	0.5	-0.8	-1.8	-4.9	-7.7	-12.3	-18.9	-30	-36	-36	-36	-36	-36	-36	-26.7	-17.1	-9.3	-4.9	-1.4	0.4	1.5	1.8	2.4	2.2	2.5	2.2	2.2	
1.2	1.1	0.4	0.4	-1.2	-2.5	-5.5	-8.6	-14	-19.5	-31	-36	-36	-36	-36	-36	-36	-28.9	-19	-11.3	-6.3	-2.8	-0.9	0.5	1.2	1.6	1.8	2.1	2.1	2.1	
1.1	0.9	0.4	0.4	-1.4	-2.8	-5.6	-8.4	-13.4	-19	-30.5	-36	-36	-36	-36	-36	-36	-30.1	-20.2	-12.8	-7.4	-4.1	-1.6	-0.4	0.5	1.4	1.5	1.8	1.8	1.9	
0.9	0.9	0.4	-0.5	-1.5	-2.8	-5.5	-8.4	-13.2	-17.3	-27.9	-34.7	-36	-36	-36	-36	-36	-35.9	-28.8	-19.7	-13.2	-7.9	-4.6	-2.1	-0.7	0.4	0.9	1.1	1.6	1.6	1.6
0.9	0.8	0.4	-0.5	-1.5	-2.6	-5.2	-7.6	-11.3	-15.6	-24.3	-30.4	-36	-36	-36	-36	-36	-32.8	-25.8	-18.2	-12.3	-7.6	-4.3	-2.4	-0.9	0.4	0.8	0.9	1.5	1.5	1.6
0.9	0.7	0.4	-0.5	-1.2	-2.1	-4.6	-6.7	-10	-12.4	-20	-25	-33	-34.2	-36	-34.6	-32.5	-28.1	-20.9	-15.6	-10.7	-6.6	-4.3	-2.5	-0.9	-0.5	0.5	0.8	1.2	1.4	1.5
0.9	0.8	0.4	0.4	-0.9	-1.5	-3.8	-5.6	-8.7	-9.8	-16	-19	-26.5	-28.2	-31.3	-28.4	-27.2	-21	-16.5	-12.4	-8.9	-5.7	-3.8	-2.1	-0.8	-0.5	0.5	0.8	1.2	1.2	1.4
0.9	0.8	0.4	0.4	-0.8	-0.9	-3.2	-4.5	-6.6	-8.2	-12.1	-14.8	-19.2	-20.3	-23.3	-20.9	-19.2	-16.6	-12.8	-10	-7.2	-4.5	-3.2	-1.8	-0.8	-0.5	0.5	0.8	1.1	1.2	1.6
0.9	0.8	0.4	0.4	-0.7	-0.7	-2.4	-3.2	-5.2	-6	-9.1	-10	-13.9	-15.5	-15.8	-15.2	-13.9	-12.3	-10	-7.6	-5.6	-3.6	-2.5	-1.4	-0.7	0.4	0.5	0.8	1.1	1.2	1.2
0.9	0.9	0.4	0.5	-0.5	-0.5	-1.6	-2.5	-3.8	-4.3	-6.2	-7.3	-9.7	-10	-11.1	-10	-9.7	-8.3	-7.2	-5.3	-4.2	-2.6	-1.9	-0.9	-0.5	0.4	0.5	0.8	1.1	1.1	1.2
0.9	1	0.5	0.8	0.4	0.4	-0.9	-1.5	-2.2	-2.2	-4.3	-4.6	-6.7	-7.2	-7.7	-7	-6.7	-5.7	-4.9	-4.1	-2.9	-1.8	-1.2	-0.7	-0.4	0.4	0.7	0.8	1.1	1.1	1.2
1.1	1.1	0.8	0.9	0.4	0.7	-0.7	-0.8	-1.5	-1.6	-2.6	-3.3	-4.5	-4.9	-4.6	-4.3	-4.5	-3.9	-3.3	-2.4	-1.9	-1.1	-0.8	0.5	0.4	0.4	0.8	0.9	1.1	1.1	1.2
1.1	1.1	0.8	0.8	0.4	0.4	-0.5	-0.5	-0.8	-0.8	-1.6	-2.1	-2.8	-3.6	-2.8	-2.9	-2.5	-2.5	-1.9	-1.6	-1.1	-0.5	-0.5	0.4	0.4	0.7	0.9	0.9	1.1	1.2	1.2
1.1	1	0.9	0.8	0.7	0.5	0.4	0.4	-0.5	-0.5	-0.8	-0.9	-1.6	-1.6	-1.8	-1.6	-1.5	-1.2	-0.9	-0.8	-0.5	-0.5	0.4	0.4	0.7	0.8	0.9	1	1.2	1.2	1.2
1.2	1.1	1	0.9	0.8	0.5	0.5	0.5	0.4	0.4	-0.5	-0.7	-0.8	-0.7	-0.8	-0.7	-0.8	-0.5	-0.5	-0.5	0.4	0.4	0.5	0.8	0.8	1.1	1.1	1.2	1.2	1.2	1.2
1.2	1.1	1.1	0.9	0.9	0.8	0.8	0.5	0.4	0.4	0.4	0.4	-0.5	-0.5	-0.5	-0.5	-0.5	-0.5	-0.4	0.4	0.4	0.4	0.7	0.7	0.9	0.9	1.1	1.1	1.2	1.2	1.2
1.2	1.2	1.2	0.9	0.9	0.9	0.9	0.8	0.7	0.7	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.4	0.5	0.7	0.8	0.8	0.9	1.1	1.1	1.2	1.2	1.2	1.2	1.2

Test Set: 81mm Mortar

2.2	2.2	2.4	2.4	2.5	2.5	2.6	2.6	2.6	2.6	2.6	2.8	2.8	2.9	3.1	3.1	3.2	3.3	3.2	3.3	3.2	3.1	3.1	2.9	2.9	2.8	2.6	2.6	2.5	2.4	2.2		
2.4	2.4	2.5	2.6	2.6	2.6	2.6	2.6	2.8	2.8	2.9	2.9	3.1	3.2	3.2	3.3	3.5	3.6	3.6	3.6	3.5	3.5	3.3	3.3	3.2	3.1	2.9	2.8	2.6	2.5	2.4		
2.5	2.5	2.6	2.6	2.8	2.8	2.8	2.8	2.9	2.9	2.9	3.1	3.2	3.3	3.5	3.8	3.9	3.9	4.1	3.9	3.9	3.9	3.8	3.6	3.5	3.3	3.2	3.1	2.9	2.6	2.5		
2.6	2.6	2.8	2.8	2.9	2.9	2.9	2.9	2.9	2.9	3.1	3.2	3.3	3.5	3.8	4.1	4.3	4.3	4.5	4.6	4.3	4.3	4.2	4.1	3.9	3.8	3.5	3.3	3.1	2.9	2.6		
2.8	2.8	3.1	3.1	3.1	3.1	3.1	3	2.9	2.9	3.1	3.2	3.3	3.8	4.1	4.3	4.8	4.9	5	5	4.9	5	4.8	4.5	4.3	4.2	3.9	3.6	3.3	3.2	2.9		
2.9	3.1	3.2	3.2	3.2	3.2	3.1	3.1	2.8	2.8	2.8	2.9	3.2	3.8	4.3	4.8	5.3	5.6	5.7	5.9	5.5	5.7	5.3	5	4.8	4.6	4.3	4.1	3.8	3.5	3.2		
3.1	3.2	3.3	3.3	3.3	3.2	3.2	2.9	2.6	2.4	2.4	2.4	2.9	3.5	4.3	5.2	5.7	6.2	6.6	6.7	6.5	6.6	6	5.9	5.3	5.2	4.8	4.5	4.1	3.8	3.3		
3.2	3.3	3.5	3.5	3.5	3.3	3.2	2.8	2.4	1.9	1.6	1.6	1.8	2.6	3.9	5	6	6.7	7.2	7.4	7.2	7.3	6.7	6.5	6	5.6	5.2	4.8	4.3	4.1	3.6		
3.5	3.6	3.6	3.8	3.6	3.3	3.1	2.5	1.9	0.8	0.4	-0.5	-0.4	0.7	2.4	4.2	5.6	7.2	7.9	8.4	8.3	8.4	7.9	7.3	6.7	6.3	5.7	5.3	4.8	4.3	3.9		
3.6	3.9	3.9	3.9	3.8	3.5	3.1	2.2	1.2	-0.8	-1.6	-3.1	-3.5	-3.1	-0.9	1.8	4.2	6.9	8.4	9.4	9.6	9.7	8.9	8.4	7.6	7.2	6.3	5.9	5.2	4.8	4.1		
3.9	4.1	4.2	4.2	3.9	3.6	3.1	2.1	0.8	-1.5	-4.1	-6.3	-8.7	-8	-6	-1.6	2.2	6.7	9.1	10.8	11	11	10.1	9.7	8.4	7.7	6.9	6.3	5.6	5.2	4.3		
4.1	4.2	4.3	4.3	4.2	3.9	3.2	1.9	0.4	-2.9	-6.7	-10.3	-13.7	-14.1	-11.8	-5.6	0.7	7.7	11	13.5	12.8	12.7	11.7	10.7	9.3	8.6	7.6	6.9	6	5.5	4.6		
4.3	4.5	4.6	4.6	4.5	4.2	3.3	2.1	0.5	-3.8	-8.4	-13.1	-17.8	-19.6	-16.1	-7.2	0.9	10.3	13.7	15.8	15.1	14.7	13	11.8	10.1	9.4	8.3	7.3	6.6	5.9	4.9		
4.5	4.6	4.9	4.9	4.8	4.5	3.8	2.2	0.7	-3.6	-8.9	-13.8	-19	-20.2	-15.9	-5	4.9	14.8	17.3	18.8	17.8	16.8	14.5	13.1	11.1	10.3	8.9	7.9	7	6.2	5.2		
4.6	4.9	5	5.2	5.2	4.8	4.1	2.8	0.4	-2.8	-7.3	-12	-16.2	-16.4	-10.8	0.8	11.4	19.6	21	22.4	20.3	18.9	16.4	14.2	12	11	9.6	8.4	7.4	6.6	5.5		
4.8	5	5.3	5.5	5.3	5	4.3	3.2	0.9	-1.9	-5.6	-9.6	-12.1	-9.1	-3.1	8.7	16.6	23.8	24	24.1	22.3	19.9	17.5	15.2	13	11.5	10.1	8.9	7.9	6.7	5.6		
4.9	5.2	5.3	5.5	5.3	5	4.3	3.2	1.1	-1.6	-4.9	-7.6	-8.6	-4.2	2.4	-13	20.5	24.8	26	25	23.1	20.5	18.2	15.8	13.4	12	10.6	9.1	8.2	7.2	5.9		
4.9	5.2	5.3	5.5	5.2	4.6	4.1	2.6	0.4	-2.8	-6.3	-9.1	-9.6	-6.6	0.4	-10	17.8	21.2	23.8	23.7	22.4	20.2	18.2	15.9	13.5	12.3	10.8	9.4	8.3	7.2	6.2		
4.9	5.2	5.3	5.3	4.8	4.2	3.3	1.2	-1.5	-6.6	-9.8	-16.5	-17.5	-15.9	-13.8	-3.5	0.5	13.4	18.2	19.6	20.3	19	17.8	15.8	13.8	12.4	11	9.6	8.4	7.3	6.2		
4.9	5	5	4.9	4.2	3.3	2.1	-0.7	-4.5	-10.7	-15.8	-26.1	-30.5	-34	-30.9	-26.8	-10	-0.7	10	15.5	17.5	17.1	16.8	15.2	13.7	12.4	11	9.6	8.4	7.4	6.3		
4.8	4.9	4.8	4.3	3.5	2.2	0.7	-2.9	-8	-15.6	-23.8	-34.2	-36	-36	-36	-36	-30.4	-13.9	0.5	9.1	14.1	15.2	15.5	14.4	13.4	12	10.7	8.4	9.4	7.4	6.3		
4.6	4.6	4.3	3.9	2.6	0.5	-1.1	-5.2	-12.8	-21	-30.9	-36	-36	-36	-36	-36	-36	-24.8	-7.7	4.2	10	12.8	13.9	13.5	12.8	11.7	10.6	9.3	8.4	7.3	6.2		
4.3	4.3	4.1	3.3	1.6	-0.7	-3.1	-7.7	-15.8	-26.3	-36	-36	-36	-36	-36	-36	-36	-31.8	-14	0.5	7.2	10.8	12.8	12.7	12.1	11.3	10.1	9	8.2	7.2	6.2		
4.2	4.1	3.6	2.5	0.7	-2	-5	-10	-19.7	-30.6	-36	-36	-36	-36	-36	-36	-36	-35.7	-18.3	-2.2	5.2	9.4	11.5	11.8	11.4	10.7	9.8	8.9	8	7	6.2		
3.9	3.8	3.2	2.1	-1.4	-3.3	-7	-12.5	-23.3	-34.2	-36	-36	-36	-36	-36	-36	-36	-20.2	-4.2	3.5	8	10.3	11	10.7	10.1	9.4	8.4	7.7	6.9	6			
3.8	3.5	2.9	1.8	-1.7	-3.8	-8.3	-14.2	-26.1	-35.7	-36	-36	-36	-36	-36	-36	-36	-23.1	-6	1.8	6.7	9.4	10	10.1	9.6	8.9	8	7.4	6.6	5.9			
3.5	3.2	2.5	1.2	-1.4	-4.6	-9.1	-15.5	-28.1	-36	-36	-36	-36	-36	-36	-36	-36	-25	-7.6	0.4	5.3	8.3	9.1	9.4	9	8.6	7.7	7.2	6.9	5.6			
3.3	3.1	2.2	1.9	-1.6	-5	-9.6	-15.8	-28.4	-36	-36	-36	-36	-36	-36	-36	-36	-27.1	-9.7	-1.2	3.9	7.2	8.2	8.7	8.3	7.9	7.2	6.7	6.2	5.5			
3.3	2.8	2.1	1.8	-1.8	-4.9	-9.8	-15.8	-28.4	-36	-36	-36	-36	-36	-36	-36	-36	-26.3	-10.3	-2.4	3.2	6.2	7.2	7.9	7.6	7.4	6.7	6.5	5.7	5.3			
3.1	2.6	1.6	1.7	-1.9	-4.8	-9.4	-14.8	-26.8	-35.4	-36	-36	-36	-36	-36	-36	-36	-24.6	-9.8	-2.6	2.4	5.3	6.5	7.2	7	6.9	6.5	6	5.5	5			
2.9	2.5	1.6	1.7	-1.6	-4.3	-8.7	-12.8	-23	-31.5	-36	-36	-36	-36	-36	-36	-36	-32.9	-21.2	-8.9	-2.6	2.1	4.6	6	6.5	6.6	6.2	6	5.6	5.2	4.8		
2.9	2.5	1.8	1.8	-1.1	-3.3	-6.9	-10.6	-18.9	-26.5	-25.2	-36	-36	-36	-36	-36	-36	-26.4	-17.2	-7.2	-2.1	2.1	4.2	5.3	5.9	6	5.9	5.6	5.3	4.9	4.5		
2.8	2.5	1.9	1.1	-0.7	-2.4	-5.3	-7.9	-14.4	-19.5	-28.1	-31.3	-36	-36	-36	-36	-36	-34.3	-28.5	-18.9	-12.3	-5.6	-1.2	2.1	3.8	4.9	5.5	5.5	5.5	5.3	5	4.6	4.3
2.8	2.5	2.1	1.4	-0.4	-1.2	-3.3	-5.5	-9.6	-13.5	-19	-21	-29.2	-31.5	-28.4	-24.6	-19.2	-13	-8.3	-3.9	-0.7	2.1	3.5	4.5	4.9	5.2	5	4.9	4.6	4.3	4.1		
2.8	2.5	2.1	1.6	0.7	-0.5	-1.8	-3.2	-6	-8.7	-11.8	-12.8	-16.9	-18.2	-17.5	-13.5	-11.1	-8.4	-5	-1.6	0.5	2.4	3.5	4.2	4.6	4.6	4.6	4.5	4.3	4.1	3.9		

Test Set: 105mm Artillery

3.1	3.2	3.8	3.9	4.3	4.5	5.3	5.5	6.3	6.5	7.3	7.4	7.9	7.9	7.9	8	7.9	7.4	7.4	6.6	6.7	5.7	5.7	4.9	4.6	4.1	3.8	3.3	3.3	2.9	2.8		
3.2	3.5	4.1	4.3	4.8	5	5.9	6.3	7.2	7.7	8.7	8.7	9.7	9.6	9.6	9.1	8.7	8.9	7.9	7.7	6.7	6.6	5.6	5.2	4.5	4.2	3.8	3.5	3.2	2.9			
3.3	3.6	4.3	4.9	5.3	5.7	6.6	7.2	8.2	8.9	10.3	10.4	11.1	11.3	10.9	11.4	10.6	10.7	10.1	9.3	8.7	7.9	7.4	6.6	5.6	5.2	4.6	4.2	3.8	3.5	3.2		
3.6	3.9	4.6	5.3	5.7	6.6	7.3	8.4	9.1	10.7	11.5	12.4	13.2	13.5	13.9	13.7	13.2	12.8	11.7	11.3	10.4	9.1	8.4	7.4	6.5	5.6	5	4.5	4.2	3.8	3.3		
3.9	4.2	5	5.6	6.2	7.4	8.4	9.4	10.4	12.1	13.7	14.9	16	16.8	17.3	16.6	16.8	15.6	13.9	13.4	12.1	10.6	9.6	8	7.2	6.3	5.5	4.9	4.5	3.9	3.5		
4.1	4.5	5.3	6	6.6	8	9.4	10.7	11.7	13.9	15.8	18.1	19.2	21.3	20.9	21.2	20.2	18.6	17.3	15.4	14.1	12.1	10.8	9.1	7.9	6.7	6	5.3	4.8	4.2	3.8		
4.2	4.6	5.6	6.6	7	8.9	10.3	11.8	13.9	15.9	17.3	21.6	23.6	26.3	26	25.8	25.5	22	20.7	17.9	16.1	13.8	12.1	10.1	8.7	7.4	6.6	5.7	5.2	4.5	3.9		
4.3	4.9	5.7	6.7	7.4	9.1	10.7	12.5	14.9	18.1	21.9	25.7	27.8	31.1	31.1	31.9	30.4	26.8	24	21	18.2	15.4	13.4	11	9	8	7	6.2	5.5	4.8	4.1		
4.3	4.9	5.9	6.9	7.6	9.6	11	13.5	16.4	20.2	24.6	30.8	33.9	36	36	36	33.6	31.3	27.2	24.3	20.5	16.9	14.5	12	10.1	8.7	7.3	6.5	5.6	4.9	4.2		
4.5	4.9	5.7	6.9	7.6	9.6	11	13.5	16.5	20.9	26	32.5	36	36	36	36	33.9	29.2	26.1	21.9	18.2	15.4	12.7	10.7	8.9	7.6	6.7	5.9	5	4.3			
4.3	4.9	5.6	6.7	7.4	9.1	10.4	12.5	15.5	19.7	24.8	30.5	36	36	36	36	36	34.5	31.3	26.8	22.9	18.9	15.9	13.1	11	9	7.9	6.9	6	5.2	4.5		
4.3	4.6	5.3	6.3	6.9	8	9.4	10.8	13.5	15.6	20.7	24.6	34.2	33.7	36	35.9	25.3	31.6	30.5	24.4	22	17.6	15.8	13	11	9.1	7.9	6.9	6	5.3	4.5		
4.1	4.5	4.9	5.7	6.2	6.9	7.6	8.4	8.9	8.6	10.7	6.7	11.8	18.9	21.4	21.3	23.1	22	24.7	21.2	19.7	15.2	14.5	12.4	10.7	8.9	7.7	6.7	6	5.2	4.5		
3.8	4.1	4.3	4.9	5.2	5	5.5	4.5	3.6	0.5	-0.5	-5.3	-7.9	-7.7	-14.5	-7.2	3.6	7.6	14.8	13.8	15.2	13.8	13	10.8	10	8.3	7.3	6.5	5.7	5	4.5		
3.5	3.8	3.9	4.2	4.2	3.2	2.9	0.9	-2.1	-6.2	-14	-24	-36	-36	-36	-36	-36	-34.5	-23.4	-8.6	1.2	5.7	10.4	10.6	10.3	9.7	9	7.7	6.9	6.2	5.6	4.9	4.3
3.3	3.3	3.1	3.3	3.1	1.5	0.4	-2.9	-9	-15.1	-26.4	-36	-36	-36	-36	-36	-36	-36	-27.2	-11.8	-3.5	3.9	5.9	7.3	7.6	7.7	6.7	6.3	5.7	5.2	4.6	4.2	
2.8	3.1	2.5	2.5	2.1	-0.8	-1.8	-7	-17.1	-24.3	-36	-36	-36	-36	-36	-36	-36	-36	-24.8	-13.5	-3.1	1.6	4.3	5.5	6.2	5.7	5.6	5.2	4.9	4.5	3.9		
2.6	2.6	2.1	1.8	1.1	-2.2	-4.2	-9.4	-19	-29.2	-36	-36	-36	-36	-36	-36	-36	-36	-33.9	-19.7	-8.9	-2.4	1.5	3.3	4.9	4.6	4.9	4.6	4.5	4.2	3.8		
2.4	2.2	1.8	1.1	0.4	-3.8	-5.3	-11.3	-21.3	-32.2	-36	-36	-36	-36	-36	-36	-36	-36	-36	-27.7	-13.8	-6.5	-0.7	1.4	3.2	3.8	4.2	4.1	4.1	3.8	3.6		
2.2	1.9	1.2	0.7	-0.7	-4.5	-7.6	-12.8	-23	-33.5	-36	-36	-36	-36	-36	-36	-36	-36	-36	-31.6	-17	-9	-3.8	-0.5	1.8	2.6	3.3	3.3	3.8	3.3	3.3		
2.1	1.6	0.9	0.4	-0.9	-4.9	-8	-13	-23.6	-33.3	-36	-36	-36	-36	-36	-36	-36	-36	-36	-32.9	-18.3	-11	-4.3	-1.8	0.8	1.9	3.1	2.9	3.2	3.2	3.2		
1.8	1.5	0.8	-0.5	-0.9	-4.9	-8	-12.5	-22.3	-31.2	-36	-36	-36	-36	-36	-36	-36	-36	-36	-31.8	-17.9	-11.3	-5.7	-2.4	0.4	1.2	2.2	2.6	2.9	2.8	2.9		
1.6	1.4	0.8	-0.5	-0.9	-4.6	-7.7	-11	-19	-26.7	-36	-36	-36	-36	-36	-36	-36	-36	-36	-28.2	-16.9	-10.4	-6	-2.5	-0.5	0.9	1.9	2.4	2.6	2.6	2.8		
1.6	1.2	0.8	0.4	-0.7	-3.8	-6.6	-8.7	-15.8	-21.9	-30.6	-33.9	-36	-36	-36	-36	-36	-36	-36	-32.8	-23	-15.4	-9.1	-5.5	-2.4	-0.5	0.8	1.6	2.1	2.4	2.4	2.6	
1.6	1.4	0.7	0.4	-0.5	-3.2	-5	-7.3	-11.5	-17.1	-22.7	-26.3	-33.9	-36	-36	-36	-36	-36	-36	-28.7	-25.3	-17.5	-11.8	-6.7	-4.2	-1.9	-0.5	0.8	1.6	1.9	2.2	2.2	2.5
1.6	1.4	0.8	0.7	-0.5	-2.4	-3.8	-5.6	-9.1	-12.1	-17.1	-19.2	-24.7	-28.1	-29.2	-27.7	-27.1	-21.4	-17.3	-12.5	-8.7	-5.5	-3.2	-1.5	-0.4	0.9	1.6	1.8	2.1	2.2	2.4		
1.6	1.5	0.9	0.9	-0.4	-1.6	-2.6	-3.8	-6.5	-8.4	-11.5	-12.1	-16.8	-18.2	-19	-16.8	-17.3	-14.7	-12.3	-9.1	-6.2	-3.8	-2.2	-0.8	0.4	0.9	1.6	1.8	2.1	2.1	2.2		
1.6	1.6	1.1	1.1	0.4	-0.7	-1.6	-2.2	-4.2	-5.5	-7.7	-7.6	-11	-12.4	-12.8	-12.3	-11	-9.8	-8.4	-6.5	-3.9	-2.6	-1.2	-0.5	0.5	1.1	1.6	1.6	2	2.1	2.1		
1.8	1.6	1.2	1.5	0.5	-0.5	-0.7	-0.8	-2.4	-3.2	-4.8	-4.5	-6.7	-7.9	-8.2	-7.4	-6.5	-5.9	-4.9	-3.9	-2.2	-1.4	-0.7	0.4	0.8	1.2	1.6	1.6	1.9	1.9	2.1		
1.8	1.6	1.4	1.6	0.9	0.5	-0.5	-0.5	-1.4	-1.5	-2.9	-2.4	-4.2	-4.5	-5.5	-4.3	-3.8	-3.5	-3.2	-1.9	-1.1	-0.5	0.4	0.7	1.1	1.4	1.6	1.8	1.9	1.9	2.1		
1.8	1.8	1.5	1.8	1.2	1.1	0.4	0.4	-0.5	-0.7	-1.5	-0.7	-2.2	-2.6	-2.6	-2.6	-2.2	-1.9	-1.4	-1.1	-0.5	0.4	0.5	0.9	1.4	1.5	1.8	1.8	1.9	1.9	2.1		
1.8	1.8	1.6	2.1	1.4	1.2	0.9	1	0.4	0.4	0.5	0.4	-0.9	-1.4	-1.5	-1.4	-0.9	-0.8	-0.5	-0.5	0.4	0.5	0.9	1.1	1.5	1.6	1.8	1.8	1.9	1.9	1.9		
1.8	1.9	1.6	2.1	1.5	1.5	1.1	1.5	0.7	0.8	0.4	0.8	-0.5	-0.7	-0.7	-0.7	-0.5	-0.5	-0.4	0.4	0.8	0.9	1.2	1.4	1.6	1.6	1.8	1.8	1.9	1.9	1.9		
1.9	1.9	1.6	2.2	1.6	1.6	1.4	1.6	1.1	1.1	0.8	1.2	0.4	0.4	0.4	0.4	0.5	0.8	0.8	1.1	1.2	1.5	1.5	1.6	1.6	1.9	1.8	1.9	1.8	1.9			
1.8	1.9	1.6	2.2	1.6	1.9	1.5	1.9	1.4	1.5	1.1	1.8	0.8	0.5	0.5	0.5	0.8	0.9	1.1	1.1	1.4	1.5	1.6	1.6	1.8	1.6	1.9	1.8	1.9	1.8	1.8		

Test Set: 105mm HEAT

APPENDIX G: SOURCE CODE (C++ NETWORK)

Copyright © 1997 Jeff May

```
#include <iostream.h>
```

```
#include <fstream.h>
```

```
#include <iomanip.h>
```

```
#include <time.h>
```

```
#include <math.h>
```

```
#include <stdlib.h>
```

```
const float STEP = .1;
```

```
const int INPUTSIZE = 1085;
```

```
const int OUTPUTSIZE = 5;
```

```
const int ROW = 35;
```

```
const int COLUMN = 31;
```

```
void displayInput (float[]);
```

```
void display(float[]);
```

```
void makeRandomWeights(float[][INPUTSIZE]);
```

```
void display (float[][INPUTSIZE]);
```

```
void mySummation(float[][INPUTSIZE], float[], float[]);
```

```
float sigmoid(float);
```

```

void buildInput(istream, float[]);

void feedForward(float[], float[][INPUTSIZE], float[], float[][INPUTSIZE], float[]);

float computeError(float[], float[], float[]);

void calcHiddenError(float[], float[][INPUTSIZE], float[], float[]);

void changeWeights(float[], float[], float[][INPUTSIZE], const int size);

```

```

main ()
{
    float weightLayer1[INPUTSIZE][INPUTSIZE] = {{0}, {0}};

    float inputLayer2[INPUTSIZE] = {0};

    float weightLayer2[OUTPUTSIZE][INPUTSIZE] = {{0},{0}};

    float outputLayer[OUTPUTSIZE] = {0};

    float t60mm[INPUTSIZE] = {0};

    float t81mm[INPUTSIZE] = {0};

    float tarty[INPUTSIZE] = {0};

    float theat[INPUTSIZE] = {0};

    float t3_5in[INPUTSIZE] = {0};

    float *ammo[OUTPUTSIZE] = {t60mm, t81mm, tarty, theat, t3_5in};

    float output60[OUTPUTSIZE] = {1,0,0,0,0};

    float output81[OUTPUTSIZE] = {0,1,0,0,0};

    float outputtarty[OUTPUTSIZE] = {0,0,1,0,0};

    float outputtheat[OUTPUTSIZE] = {0,0,0,1,0};

```

```

float output3_5in[OUTPUTSIZE] = {0,0,0,0,1};

float *expected[OUTPUTSIZE] = {output60, output81, outputarty, outputheat, output3_5in};

ifstream mortar1("60mm.dat", ios::in);

ifstream mortar2("81mm.dat", ios::in);

ifstream arty("arty.dat", ios::in);

ifstream heat("heat.dat", ios::in);

ifstream rocket("3-5in.dat", ios::in);

ofstream layer1weights("data/weight1.dat", ios::out);

ofstream layer2weights("data/weight2.dat", ios::out);

ofstream errorData("data/error.dat", ios::out);

float error1 = 0;

float totalError = 0;

const float ACCEPTABLE_ERROR = 1.0;

float outputError[OUTPUTSIZE] = {0};

float hiddenError[INPUTSIZE] = {0};

int counter = 0;

ifstream *infiles[OUTPUTSIZE] = {mortar1, mortar2, arty, heat, rocket};

for(int m = 0; m < OUTPUTSIZE; m++) {

    buildInput(infiles[m], ammo[m]);

}

```

```

makeRandomWeights(weightLayer1);

makeRandomWeights(weightLayer2);

do{

    counter = counter + 1;

    cout << "counter " << counter << endl;

    totalError = 0;

    for(int e = 0; e < OUTPUTSIZE; e++) {

        feedForward(ammo[e], weightLayer1, inputLayer2, weightLayer2, outputLayer);

        error1 = computeError(outputLayer, expected[e], outputError);

        calcHiddenError(outputError, weightLayer2, inputLayer2, hiddenError);

        changeWeights(outputError, inputLayer2, weightLayer2, OUTPUTSIZE);

        changeWeights(hiddenError, inputLayer2, weightLayer1, INPUTSIZE);

        totalError = error1 + totalError;

    }

    cout <<"totalError " << totalError << endl;

    errorData << totalError << endl;

} while(totalError > ACCEPTABLE_ERROR);

errorData.close();

for(int r = 0; r < INPUTSIZE; r++) {

    for(int c = 0; c < INPUTSIZE; c++) {

        layer1weights << weightLayer1[r][c] << " ";

```

```

    }

    layer1weights << endl;

}

layer1weights.close();


for(int rr = 0; rr < OUTPUTSIZE; rr++) {

    for(int cc = 0; cc < INPUTSIZE; cc++) {

        layer2weights << weightLayer2[rr][cc] << " ";

    }

    layer2weights << endl;

}

layer2weights.close();

return(0);

}

```

```

void feedForward(float input1[], float weight1[][INPUTSIZE], float input2[], float
weight2[][INPUTSIZE],float output[])

```

```

{

    float tempSum = 0;


    mySummation(weight1, input1, input2);

```

```

for(int j = 0; j < INPUTSIZE; j++){
    tempSum = input2[j];
    input2[j] = sigmoid(tempSum);
}

mySummation(weight2, input2, output);

tempSum = 0;

for(int l = 0; l < INPUTSIZE; l++){
    tempSum = output[l];
    output[l] = sigmoid(tempSum);
}

cout <<"outputLayer" << endl;

display(output);
}

float sigmoid(float sum){
    float answer = 0;

    answer = (1 / (1 + (exp (-sum)))));

    return(answer);
}

```

```
void makeRandomWeights(float Array[] [INPUTSIZE])
```

```
{  
    srand(time(NULL));  
    for(int r = 0; r < INPUTSIZE; r++){  
        for(int c = 0; c < INPUTSIZE; c++){  
            Array[r][c] = (1.0 * ((rand() % 3) - 1));  
        }  
    }  
}
```

```
void mySummation(float weight[][INPUTSIZE], float input[], float output[])
```

```
{  
    float temp[INPUTSIZE][INPUTSIZE];  
    int sum = 0;  
  
    for(int r = 0; r < INPUTSIZE; r++){  
        for(int c = 0; c < INPUTSIZE; c++){  
            temp[r][c] = weight[r][c] * input[c];  
        }  
    }  
}
```

```

for(int i = 0; i < INPUTSIZE; i++){
    for(int j = 0; j < INPUTSIZE; j++){
        sum = temp[i][j] + sum;
    }
    output[i] = sum;
    sum = 0;
}
}

```

```

void display(float Array[][INPUTSIZE])
{
    for(int r = 0; r < INPUTSIZE; r++){
        for(int c = 0; c < INPUTSIZE; c++){
            cout << setw(3) << Array[r][c];
        }
        cout << endl;
    }
    cout << endl;
}

```



```

void display(float Array[])
{
    for(int i = 0; i < OUTPUTSIZE; i++){
        cout << Array[i] << " ";
    }
    cout << endl;
}

```

```

void displayInput(float Array[])
{
    int column = 0;
    int end = 0;

    for (int r = 0; r < ROW; r++){
        end = column + COLUMN;
        for (; column < end; column++){
            cout << Array[column] << " ";
        }
        cout << endl;
    }
    cout << endl;
}

```

```
}
```

```
void buildInput(ifstream infile, float ammoArray[])
```

```
{
```

```
    infile.open();
```

```
    cout << "in build input" << endl;
```

```
    for(int i = 0; i < INPUTSIZE; i++) {
```

```
        infile >> ammoArray[i];
```

```
    }
```

```
    infile.close();
```

```
}
```

```
float computeError(float actual[], float expected[], float error[])
```

```
{
```

```
    float result = 0;
```

```
    float temp = 0;
```

```
    for(int i = 0; i < OUTPUTSIZE; i++) {
```

```
        error[i] = expected[i] - actual[i];
```

```
        if(error[i] < 0) {
```

```
            temp = error[i] * -1.0;
```

```
        }
```

```

    else {

        temp = error[i];

    }

    result = temp + result;

}

return(result);

}

```

```

void calcHiddenError(float outError[], float weight2[][INPUTSIZE], float input2[], float
hidden[])

```

```

{

for(int c = 0; c < INPUTSIZE; c++) {

    for(int r = 0; r < OUTPUTSIZE; r++) {

        hidden[c] = (outError[r] * weight2[r][c]) + hidden[c];

    }

}

for(int i = 0; i < INPUTSIZE; i++) {

    hidden[i] = hidden[i] * (input2[i] * (1 - input2[i]));

}

}

```

```

void changeWeights(float error[], float input[], float weight[][INPUTSIZE], const int ROW)
{
    float temp = 0;

    //cout << "in changeweights" << endl;

    for(int r = 0; r < ROW; r++) {
        for(int c = 0; c < INPUTSIZE; c++) {
            temp = STEP * error[r] * input[c];
            weight[r][c] = weight[r][c] + temp;
        }
    }
}

```

LIST OF REFERENCES

1. Young, R. Helms, L., "*Applied Geophysics and the Detection of Buried Munitions*", August 96.
2. Reference Manual, GA-72Cd Magnetic Locator, Schonstedt Instrument Company.
3. Foley, J., Gifford, M., "*Ordnance and Explosives (OE) Program Geographic Information System (GIS) and Knowledge Base (KB)*", Sanford Cohen and Associates Inc.
4. Jain, A., Mao, J., Mohiuddin, K., "*Artificial Neural Networks: A Tutorial*," Computer, Vol. 29, No. 3, March 1996, pp. 31-44.
5. Cantrell, M., *Speech Recognition Using Artificial Neural Networks*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1996.
6. Koiran, P., "*On the Complexity of Approximating Mappings Using Feedforward Networks*", Neural Networks, 1993, Vol. 6, pp.649-653.
7. Haykin, S., *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Company, New York, 1994.
8. Lippmann, R., "*An Introduction to Computing with Neural Nets*", IEEE ASSP Magazine, April 1987, pp. 4-22.
9. Hudson, P., Postma, E., "*Choosing and Using a Neural Net*", Artificial Neural Networks, Springer-Verlag, Heidelberg, Germany, 1995.
10. Dean, T., Allen, J., Aloimonos, Y., *Artificial Intelligence Theory and Practice*, Benjamin/Cummings, Redwood City, California, 1995.
11. Caudill, M., Butler, C., *Understanding Neural Networks*, MIT Press, Vol. 1, Fifth Printing, 1994.
12. Graham, P., "*ANSI Common Lisp*," Prentice Hall, Englewood Cliffs, New Jersey, 1996.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center.....2
8725 John J. Kingman Road, Ste 0944
Ft. Belvoir, VA 22060-6218
2. Dudley Knox Library2
Naval Postgraduate School
411 Dyer Rd.
Monterey, CA 93943-5101
3. Chairman, Code CS1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943
4. Dr. Nelson D. Ludlow, Code CS/Lw2
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943
5. Dr. Robert B. McGhee, Code CS/Mz1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943
6. Dr. Yukata Kanayama, Code CS/Kz.....1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

7. CPT Jeffrey A. May5
Defense Intelligence Agency
ATTN: DIAC Collateral Distribution Center
Washington, D.C. 20340
Inbound - 15 July 97
8. ECJ6-NP.....1
HQ USEUCOM
Unit 30400 Box 1000
APO AE 09128